

Runtime System

Reorder No. 0101

MICROPORT SYSTEMS

The material contained in this manual was reprinted with permission from AT&T and is comprised of excerpts from the following AT&T manuals.

*UNIX System V - Release 2.0 User Reference Manual	March 1985
†INTEL Processors - Version 1	307-605
UNIX System V - Release 2.0 User Guide	April 1984
	307-100, Issue 2
UNIX System V - Release 2.0 Administrator Guide	March 1985
INTEL Processors	307-622, Issue 1
UNIX System V - Release 2.0 Operator Guide	March 1985
INTEL Processors	307-623, Issue 1
UNIX System V - Release 2.0 Administrator Reference Manual	March 1985
INTEL Processors	307-626, Issue 1
UNIX System V - Release 2.0 Release Notes	January 1986
INTEL Processor - Version 1	307-621, Issue 2

*UNIX is a trademark of AT&T Bell Laboratories

†INTEL is a trademark of Intel Corporation

Copyright © 1984, 1985, 1986 by AT&T

All rights reserved

Printed in U.S.A.

DIABLO is a registered trademark of Xerox Corporation

UNIX is a trademark of AT&T Bell Laboratories

INTEL and **iAPX 286** is a trademark of Intel Corporation

DOCUMENTER'S WORKBENCH is a trademark of AT&T

DEC, PDP, and VAX are trademarks of Digital Equipment Corporation

HP is a trademark of Hewlett-Packard, Inc.

VAX is a trademark of Digital Equipment Corporation

SYSVISION is a trademark of Microport Systems, Inc.

(Contents of SysVision provided by TaskForce)

INFORMIX is a trademark of Informix Corporation

UNIFY is a trademark of Unify Corporation

Preface

You have in your hands a revolutionary new product, Microport's UNIX* System V for the IBM PC-AT. The System V/AT product, derived from the certified port of UNIX System V Release 2 for the iAPX286 Processor Version 1, brings the UNIX system as developed by AT&T Bell Laboratories to the world of desktop computing. Because of the elegance of its design, and because of all it can do, the UNIX system has gained wide popularity since it was introduced in the late 1960s. Now you are about to implement it on your AT-compatible system.

This manual is intended for a wide range of potential UNIX users, from novice to expert. You will find that learning the UNIX system requires some thought and time, and that you will be rewarded for your efforts with power and flexibility that far surpasses other operating systems.

How to Use This Manual

The beginning user should probably start with Chapter 1, "Introduction and Product Overview." This provides the novice user with a brief description of the UNIX system, followed by a listing of feature updates and improvements, which an experienced user will find helpful. Chapter 4, "UNIX System Capabilities," expands on the information in Chapter 1, as does Chapter 6, "Using the File System." Then, to install the package on your hardware, refer to Chapter 2, "Installation Instructions." To perform administrative tasks, Chapter 3, "SysVision," provides menus and forms designed to aid novice and experienced users. Other chapter topics are specific to particular needs, including using the vi editor, setting up the line printer spooler, communicating with other UNIX system users, and adding device drivers.

For reference information on all commands, including communications (1C) and systems maintenance commands (1M) and for references to the special files interface to peripheral devices consult the last two sections of the manual, section 1 (Commands) and section 7 (Special Files).

Note that sections from the UNIX System V Release 2.0 Programmer Reference Manual, that is, section 2 (System Calls), section 3 (Library Functions), section 4 (File Formats) and section 5 (Miscellany) are found in the Software Development System manual.

For system administration considerations, consult Chapter 3, "SysVision," and Chapter 5, "Single User and Multiuser," and Chapters 8 and 9, "Administrative Advice" and "Fsck." Finally, in order to find out about technical differences between System V/AT and other UNIX systems, consult Appendix C to Chapter 1, "Product Overview."

Preface

Chapter 1: Introduction and Product Overview

This chapter acquaints you with System V/AT and explains how the UNIX system works. Appendix C also gives technical details of the product, which is derived from UNIX System V release 2, iAPX286 Processor Version 1, and includes updates and improvements over release 1, and features of System V/AT in comparison with other flavors of the UNIX system.

Chapter 2: Installation Information

This chapter takes you step by step through the installation of System V/AT.

Chapter 3: SysVision

This chapter presents you with menus and forms to aid you in using the UNIX system, designed to help novice UNIX system users with system administration tasks.

Chapter 4: UNIX System Capabilities

This chapter highlights UNIX system capabilities such as command execution, text editing, electronic communication, programming, and aids to software development.

Chapter 5: Single User and Multiuser

This chapter gives procedures and examples for changing between single and multiuser, saving and restoring files, bringing down the system, and restoring after a crash.

Chapter 6: Using the File System

This chapter introduces the file system and explains how you can use it to organize information, and describes commands for storing and retrieving this information.

Chapter 7: Screen Editor Tutorial

This chapter teaches you how to use the vi text editor to create and modify text on the video display terminal or monitor.

Chapter 8: Administrative Advice

This chapter contains helpful advice and suggestions regarding system administration, including system tuning, allocation resources, and trouble shooting.

Chapter 9: Fck (File System Check Utility)

This chapter describes the file system check program of the UNIX system. Fck audits and interactively repairs inconsistency in the file system.

Chapter 10: LP Spooling

This chapter defines the line printer (LP) spooling system package and describes LP administration functions.

Chapter 11: Communication Tutorial

This chapter teaches you how to send information to others, and receive information from others, whether they are working on your UNIX system or a different one.

Chapter 12: UUCP (Unix to Unix Copy)

This chapter describes how a UUCP network is set up, the format of the control file, and administrative procedures.

Chapter 13: Adding Device Drives Using Link Kit

This chapter contains the rules and procedures that should be followed for writing device drivers for Microport's System V/AT UNIX, in order to add peripheral devices to the system.

Section 1: Commands and Application Programs

References for all section (1) commands, including (1C) and (1M).

Section 7: Special Files

References for the special files interface to peripheral devices.

QUICK REFERENCE GUIDE

INTRODUCTION & OVERVIEW C-1

INSTALLATION INFORMATION C-2

SYSVISION C-3

UNIX SYSTEM CAPABILITIES C-4

SINGLE USER AND MULTIUSER C-5

USING THE FILE SYSTEM C-6

SCREEN EDITOR TUTORIAL C-7

ADMINISTRATIVE ADVICE C-8

FCK C-9

LP SPOOLING C-10

COMMUNICATION TUTORIAL C-11

UUCP C-12

ADDING DEVICE DRIVERS C-13

COMMANDS S-1

SPECIAL FILES S-7

Table of Contents

TITLE	CHAPTER
INTRODUCTION & PRODUCT OVERVIEW.....	1
INSTALLATION INFORMATION.....	2
SYSVISION.....	3
UNIX SYSTEM CAPABILITIES.....	4
SINGLE USER AND MULTUSER.....	5
USING THE FILE SYSTEM.....	6
SCREEN EDITOR TUTORIAL.....	7
ADMINISTRATIVE ADVICE.....	8
FCK.....	9
LP SPOOLING.....	10
COMMUNICATION TUTORIAL.....	11
UUCP.....	12
ADDING DEVICE DRIVERS USING LINK KIT.....	13
SECTION	
COMMANDS, APPLICATION PROGRAMS (1), (1C) & (1M).....	1
SPECIAL FILES (7).....	7

Chapter 1

Introduction and Product Overview

Welcome to Microport System V/AT. In this introductory chapter, we explain what System V/AT is, and explore some of its features. We also describe the System V/AT Runtime manual and handbook that you get with the system, and show you how to use them.

This chapter is divided into several parts, described as follows:

- The first part, *What is System V/AT?*, describes some of the features of Microport System V/AT, including a brief summary of the system manuals, and explains how to use your manuals.
- The second part, *How to Use the UNIX System*, explains what the UNIX operating system is, and how you can use it to communicate with your computer.
- The third part is contained in several *appendices*, which contain valuable information specific to Microport System V/AT.

The more experienced UNIX user may want to proceed directly to the appendices, which are described briefly below:

Appendix A tells you about some useful System V/AT features, such as *virtual consoles*, not available in other versions of UNIX.

Appendix B explains UNIX documentation conventions.

Appendix C contains a detailed summary of improvements to the generic System V release that help the experienced UNIX system user to differentiate System V/AT UNIX from other "flavors" of the UNIX system.

We strongly recommend that all novice UNIX system users, or those unfamiliar with Microport System V/AT features, take time to read this introductory chapter and its appendices.

WHAT IS SYSTEM V/AT?

System V/AT is a version of the UNIX system specifically designed for the IBM PC/AT desktop computer. System V/AT can share a hard disk with DOS, giving you two independent operating systems to work with. System V/AT features include:

- multi-user and multi-tasking capabilities
- all standard UNIX System V capabilities
- user friendly features such as SystemVision (a collection of menus and forms designed to help novice users with System Administration tasks)
- ability to co-reside on a fixed disk with DOS
- utilities to read and write DOS files
- Berkeley C Shell
- ability to run sophisticated data base applications, such as Informix and Unify
- ability to run additional applications, such as office automation and word processing packages

How To Use Your System V/AT Documentation

Microport System V/AT is divided into three parts. The main part, the Runtime System, comes with a manual and a handbook. Two optional parts of the package, the Software Development System and the Text Preparation System, each come with their own documents. A brief description of each document follows:

The Runtime System

The Runtime manual is really two separate books combined into one; a User's manual and a Command Reference manual. The User part of the manual teaches you how to install and use System V/AT. The Command Reference part is actually two sections at the back of the manual, Section 1 and Section 7. Section 1, *Commands*, provides reference material for UNIX system commands; and Section 7, *Special Files Interfaces to Peripheral Devices*, tells you how to access devices such as the floppy drive, or how to install a modem.

To aid the novice user, we include an introductory user handbook with each Runtime system. The handbook, *Microport System V Made Easy: Using the UNIX Operating System*, teaches the basic system utility commands needed to get you started on UNIX.

The Software Development Manuals

The Software Development manuals explain the various system tools and how to use them. In Volume I, you will find general material relating to the C compiler, the *ld* linker, *make* utility, and Source Code Control System. Volume II provides reference material for the system calls, libraries, and file formats.

The Text Preparation System

The Text Preparation System manual teaches you text preparation tools, including *nroff* and *troff* formatting and typesetting commands.

A note to the reader: You may notice throughout the manuals that we tend to use the words *the UNIX system* and *Microport System V/AT* almost interchangeably. This is because Microport System V/AT is our version of the UNIX operating system, so that information pertaining to one is usually true for both.

HOW TO USE THE UNIX SYSTEM

The UNIX system is a set of programs, called software, that acts as the link between a computer and you, its user. The UNIX system is designed to control the computer on which it is running so the computer can operate efficiently and smoothly and to provide you with an uncomplicated, efficient, and flexible computing environment.

UNIX system software does three things:

- It controls the computer,
- It acts as an interpreter between you and the computer, and
- It provides a package of programs or tools that allows you to do your work.

The UNIX system software that controls the computer is referred to as *the operating system*. The operating system coordinates all the details of the computer's internals, such as allocating system resources and making the computer available for general purposes. The nucleus of this operating system is called the kernel.

In the UNIX system, the software that acts as a liaison between you and the computer is called the *shell*. The shell interprets your requests and, if valid, retrieves programs from the computer's memory and executes them.

The UNIX system software that allows you to do your work includes programs and packages of programs, called *tools*, for electronic communication, for creating and changing text, and for writing programs and developing software tools.

PRODUCT OVERVIEW

Put simply, this package of services and utilities called the UNIX system offers:

- A *general purpose* system that makes the resources and capabilities of the computer available to you for performing a wide variety of jobs or applications, not simply one or a few specific tasks.
- A computing environment that allows for an *interactive* method of operation so you can directly communicate with the computer and receive an immediate response to your request or message.
- A technique for sharing what the system has to offer with other users, even though you have the impression that the UNIX system is giving you its undivided attention. This is called *timesharing*. The UNIX system creates this feeling by allowing you and other users--*multiusers*--slots of computing time measured in fractions of seconds. The rapidity and effectiveness with which the UNIX system switches from working with you to working with other users makes it appear that the system is working with all users simultaneously.
- A system that provides you with the capability of executing more than one program simultaneously, this feature is called *multitasking*.

The UNIX system, like other operating systems, gives the computer on which it runs a certain profile and distinguishing capabilities. But unlike other operating systems, it is largely machine-independent; this means that the UNIX system can run on mainframe computers as well as microcomputers and minicomputers.

From your point of view, regardless of the size or type of computer you are using, your computing environment will be the same. In fact, the integrity of the computing environment offered by the UNIX system remains intact, even with the addition of optional UNIX system software packages that enhance your computing capabilities.

HOW THE UNIX SYSTEM WORKS

After reading the past few pages, you know that the UNIX system offers you a set of software that performs services—some automatically, some you must request. You also know that the system creates a certain environment in which you can use its software. But before you can make requests of the UNIX system, you need to know what it can do.

Figure 1-1 shows a set of layered circles in graduated sizes. Each circle represents specific UNIX system software, such as:

- Kernel,
- Shell,
- Programs/tools that run on command.

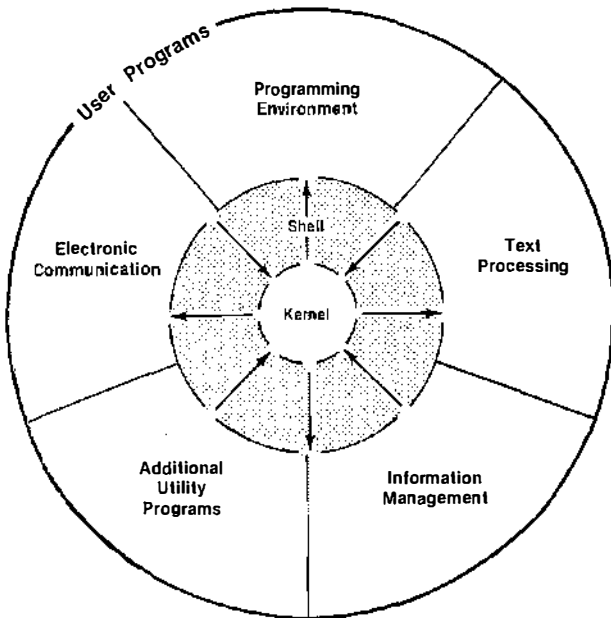


Figure 1-1. UNIX system model

You should know something about the major components of UNIX system software to communicate with the UNIX system. Therefore, the remainder of this chapter introduces you to each component: the kernel, the shell, and user programs or commands.

Kernel

The heart of the UNIX system is called the *kernel*. *Figure 1-2* gives an overview of the kernel's activities. Essentially, the kernel is software that controls access to the computer, manages the computer's memory, and allocates the computer's resources to one user, then to another. From your point of view, the kernel performs these tasks automatically. The details of how the kernel accomplishes this are hidden from you. This arrangement lets you focus on your work, not on the computer's.

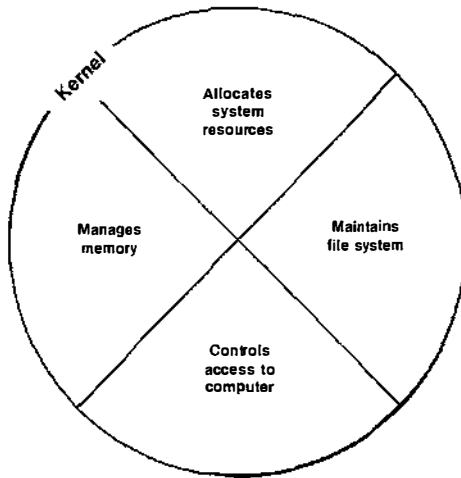


Figure 1-2. Functional view of kernel

On the other hand, you will become increasingly familiar with another feature of the kernel; this feature is referred to as the *file system*.

The file system is the cornerstone of the UNIX operating system. It provides you with a logical, straightforward way to organize, retrieve, and manage information electronically. If it were possible to see this file system, it might look like an inverted tree or organization chart made up of various types of files *Figure 1-3*. The file is the basic unit of the UNIX system and it can be any one of three types:

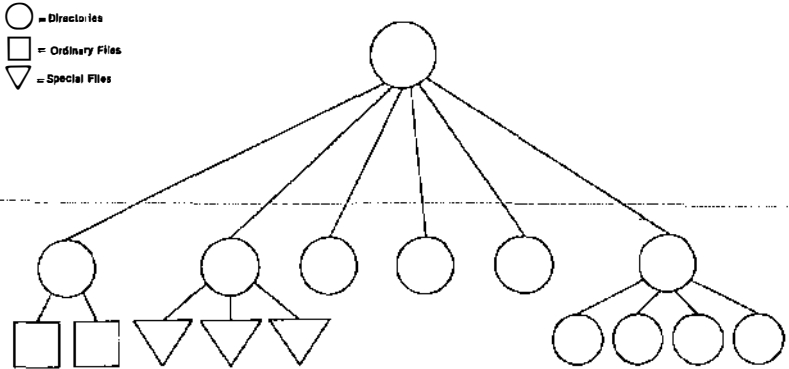


Figure 1-3. Branching directories and files give the UNIX system its treelike structure

- An *ordinary file* is simply a collection of characters. Ordinary files are used to store information. They may contain text or data for the letters or reports you type, code for the programs you write, or commands to run your programs. In the UNIX system, everything you wish to save must be written into a file.

In other words, a file is a place for you to put information for safekeeping until you need to recall or use its contents again. You can add material to or delete material from a file once you have created it, or you can remove it entirely when the file is no longer needed.

- A *directory* is a file maintained by the operating system for organizing the treelike structure of the file system. A directory contains files and other directories as designated by you. You can build a directory to hold or organize your files on the basis of some similarity or criterion, such as subject or type.

For example, a directory might hold

- Files containing memos and reports you write pertaining to a specific project or client.
- Files containing research specifications and programming source code for product development.
- Files of executable code allowing you to run your computing jobs.
- Files representing any combination of these possibilities.
- A *special file* represents a physical device, such as the terminal on which you do your computing work or a disk on which ordinary files are stored. At least one special file corresponds to each physical device supported by the UNIX system.

In some operating systems, you must define the kind of file you will be working with and then use it in a specified way. You must consider how the files are stored since they can be sequential, random-access, or binary files. To the UNIX system, however, all files are alike. This makes the UNIX system file structure easy to use. For example, you need not specify memory requirements for your files since the system automatically does this for you. Or if you or a program you write needs to access a certain device, such as a printer, you specify the device just as you would another one of your files. In the UNIX system, there is only one interface for all input from you and output to you; this simplifies your interaction with the system.

The source of the UNIX system file structure is a directory known as root, which is designated with a slash (/). All files and directories in the file system are arranged in a hierarchy under root. Root normally contains the kernel as well as links to several important system directories that are shown in *Figure 1-4*:

- | | |
|-------------|--|
| /bin | Many executable programs and utilities reside in this directory. |
| /dev | This directory contains special files that represent peripheral devices, such as the console, the line printer, user terminals, and disks. |

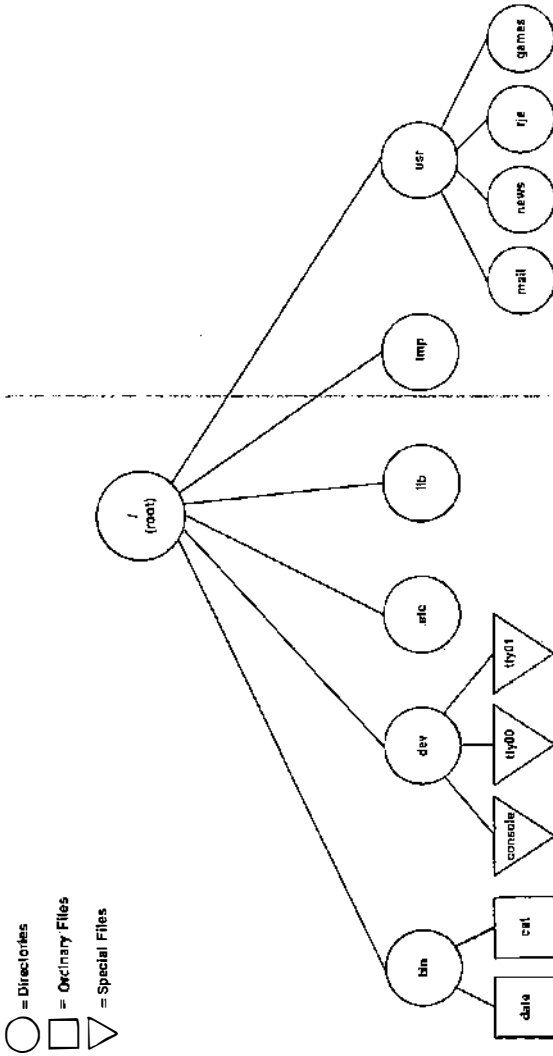


Figure 1-4. Sample of typical file system structure

PRODUCT OVERVIEW

- /etc** Programs and data files for system administration can be found in this directory.
- /lib** This directory contains available program and language libraries.
- /tmp** This directory is a place where anyone can create temporary files.
- /usr** This directory holds other directories, such as **mail** (which further holds files storing electronic mail), **news** (which contains files holding newsworthy items), and **rje** (which contains files needed to send data via something called the remote job entry communication link).

In summary, the directories and files you create comprise the portion of the file system that is structured and, for the most part, controlled by you. Other parts of the file system are provided and maintained by the operating system, such as **bin**, **dev**, **etc**, **lib**, **tmp** and **usr**, and have much the same structure on all UNIX systems.

The "Using the File System" chapter of this manual shows how to organize a file system directory structure and how to access and manipulate files. "Unix System Capabilities" gives an overview of UNIX system capabilities. The effective use of these capabilities depends on your familiarity with the file system and your ability to access information stored within it. The "Screen Editor Tutorial" is designed to teach you how to create and edit files to meet your computing and information management needs.

Shell

The shell is a unique UNIX system program or tool that is central to most of your interactions with the UNIX system. *Figure 1-1* illustrates how the shell works. The drawing shows the shell as a circle containing arrows pointing away from the kernel and the file system to the outer circle that contains programs and then back again. The arrows indicate that a two-way flow of communication is possible between you and the computer via the shell.

When you enter a request to the UNIX system by typing on the terminal keyboard, the shell translates your request into language the computer understands. If your request is valid, the computer honors it and carries out an instruction or set of instructions. Because of its job as translator, the shell is called the *command language interpreter*.

As the command language interpreter, the shell can also help you to manage information. The shell's ability to manage information stems from the design of the UNIX system. Each program in the UNIX system is designed to do one thing well. In a sense, a UNIX system program is a building block or module that you can use in tandem with other programs to create even more powerful tools.

In addition to acting as a command language interpreter, the shell is a programming language complete with variables and control flow capabilities.

A section of the next chapter describes each of the shell's capabilities. Any reference work on shell programming techniques can teach you how to use these capabilities to write simple shell programs called shell scripts and how to custom-tailor your computing environment.

Commands

A program is a set of instructions that the computer follows to do a specific job. In the UNIX system, programs that can be executed by the computer without need for translation are called *executable programs* or *commands*.

As a typical user of the UNIX system, you have many standard programs and tools available to you. If you also use the UNIX system to write programs and to design and develop software, you have system calls, subroutines, and other tools at your disposal. And you have, of course, the programs you write.

This book introduces you to approximately 40 of the most frequently used programs and tools that you will probably use on a regular basis when you interact with the UNIX system. If you need additional information on these or other standard UNIX system programs, check section 1 of the Runtime System manual.

If you want to use tools and routines that relate to programming and software development, you should consult the Software Development System manual.

The details contained in the two reference manuals may also be available via your terminal in what is called the *on-line* version of the UNIX system reference manuals. This on-line version is made up of formatted text files that look exactly like the printed pages in the manuals. You can summon pages in this electronic manual using the command **man**, which stands for **m**anual page, if the electronic version of the manuals is available on your computer. The **man** command is documented in your copy of the Runtime System manual.

What Commands Do

The outer circle of *Figure 1-1* organizes UNIX system programs and tools into general categories according to what they do. The programs and tools allow you to:

- *Process text.* This capability includes programs, such as, line and screen editors (which create and change text), a spelling checker (which locates spelling errors), and optional text formatters (which produce high-quality paper copies that are suitable for publication).
- *Manage information.* The UNIX system provides many programs that allow you to create, organize, and remove files and directories.
- *Communicate electronically.* Several programs, such as **mail**, provide you with the capability to transmit information to other users and to other UNIX systems.
- *Use a productive programming and software development environment.* A number of UNIX system programs establish a friendly programming environment by providing UNIX-to-programming-language interfaces and by supplying numerous utility programs.
- *Take advantage of additional system capabilities.* These programs include graphics, a desk calculator package, and computer games.

How Commands Execute

Figure 1-5 gives a general idea of what happens when the UNIX system executes a command.

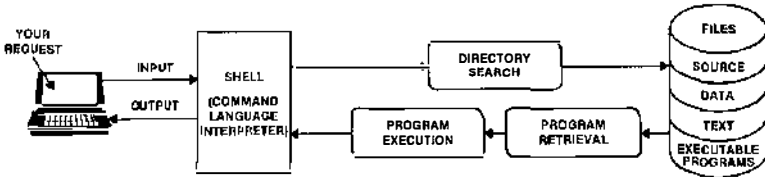


Figure 1-5. Flow of control between you and computer when you request program to run

When the shell signals it is ready to accept your request, you type in the command you wish to execute on the keyboard. The command is considered input, and the shell searches one or more directories to locate the program you specified. When the program is found, the shell brings your request to the attention of the kernel. The kernel then follows the program's instructions and executes your request. After the program runs, the shell asks you for more information or tells you it is ready for your next command.

This is how the UNIX system works when your request is in a format that the shell understands. The structure that the shell understands is called a *command line*. "Using the File System" explains what you need to know about the command line so you can request a program to run.

This chapter has outlined some basic principles of the UNIX operating system and explained how they work. The following chapters will help you begin to apply these principles according to your computing needs.

Chapter 2, "Installation Instructions," shows you how to install the UNIX system on your hard disk. Chapter 3, "System Vision," contains a set of menus and forms which you can use to set up and maintain your system.

Appendix A

USEFUL SYSTEM FEATURES

This appendix explains several implementation-specific features of Microport System V/AT, and provides valuable information for new and experienced users.

Virtual Consoles

On Microport's System V/AT, you can use the system monitor to log in to three additional pseudo-terminals, called "virtual consoles." This feature gives you the same effect as having three additional terminals sitting on your desk! This multiple login session capability is similar to Berkeley UNIX "job control" capability and allows you, for example, to keep a root login for system administration on screen-1 while editing a file in screen-2, and run your favorite UNIX application in screen-3; or, switch to screen-4 to check the spelling of a file name-you need for-your-edit.

To access the virtual consoles from your system monitor, press the SYS REQ key on your keyboard, or use the ALT-F1 through ALT-F4 keys to access Screen-1 through Screen-4 directly. Each keypress gives you a new login prompt, and the fourth keypress returns you to your original login screen. To learn more about virtual consoles, see CONSOLE(7).

System Vision

The UNIX Operating system can often be difficult for a new user to master. Our System Vision option provides users with a set of menus and forms that can be used to perform a large variety of system administrator functions quickly and conveniently. Chapter 3, System Vision, explains how you can make use of these menus to add a new user to the system, unmount, check, and remount your file systems, communicate with other system users, and for many other functions.

Berkeley C Shell

Although the standard user interface to UNIX System V is the Bourne Shell, described in Chapter 4, "UNIX System Capabilities," another popular shell is the C Shell, from the University of California at Berkeley. Because of its many capabilities, Microport has provided the C Shell as an option.

To use the C Shell, supply the pathname `/bin/csh` instead of the normal `/bin/sh` pathname when setting up a new user (see the chapter on System Vision). The syntax of the C Shell is explained in the Commands Section of this manual, "Section 1"; the reference is CSH(1).

UNIX fdisk utility and format command

Since Microport System V/AT runs in a hard disk environment often shared by the DOS operating system, Microport has provided a DOS compatible *fdisk* utility to manage the partition of the hard disk. To execute this utility to examine your hard disk partition structure, log on as root, then type "fdisk." Be aware of changing the structure of your active partition through *fdisk* without re-installing! This utility is documented in Section 1, Commands, as FDISK(1).

Another command which structures the hard disk is the System V/AT *format* command. *Format* erases any data on the disk; including the partition table created by *fdisk*. It is a low-level format of the hard disk, and is not parallel to the DOS format command, which is a "high-level" formatter, similar to the UNIX *mkfs* command.

The Nodename command

The system "nodename" is a name up to six characters long, used for *uucp* connection. In System V/AT, the default nodename is "system5". In order to display the system nodename, type "uname -a". This gives your system nodename as the second field. The system nodename is also displayed above the login prompt. Another way to display the system nodename is through the use of the *nodename* command. When logged in as "root", typing *nodename* gives you the current name, and typing *nodename xyz* changes the system nodename to *xyz*. Note that the name is changed both in memory and on the hard disk; no re-boot is required.

Appendix B

UNIX DOCUMENTATION CONVENTIONS

UNIX reference documentation, which describes the specific syntax of commands and system calls, etc., is divided into several logical parts, called *sections*. In this manual, Section 1 describes commands you will use to operate the system.

Locating Commands and System Calls

To learn how to locate a command, let's use as an example the command *cp* (*copy*), which copies one filename onto another. Any reference to the *cp* command is given as CP(1). The number (1) after the command name means that you will find a total syntactical description of *cp* in Section 1.

Similarly, a system call such as *link* would be found in Section 2, and referenced as LINK(2). This style of reference helps you to distinguish the command *link*, referred to as LINK(1), from the *link* system call, LINK(2).

(Note that Sections 2-5 are part of the Software Development manuals and are not found in the Runtime manual.)

Locating Maintenance Commands

In addition to user level commands, such as *cp*, Section 1 of this manual also contains references to the System Administration commands, such as *mount* and *umount*. These commands, sometimes referred to as *Maintenance* commands, are referenced as MOUNT(1M) and UMOUNT(1M). The (1M) commands are alphabetized together with the (1) commands in Section 1.

APPENDIX C

**UNIX SYSTEM V RELEASE 2.0
INTEL iAPX 286 PROCESSOR VERSION 1**

UPDATES AND IMPROVEMENTS

UNIX System V Release 2, iAPX 286 Processor Version 1* is being provided for the Intel iAPX 286 processor.

System V Release 2 includes feature updates and improvements in the following areas:

- Software Generation Systems
 - Commands and Utilities
 - Performance
 - Processor-dependent Features
 - Documentation
 - Record and File Locking
 - File System Hardening

The software generation system, command, and utility changes made for this release incorporate several features of the University of California at Berkeley Software Distribution of the UNIX System (BSD). Also, some Software Generation System changes have been made to take advantage of the iAPX 286 processor architecture.

* UNIX System V Release 2, iAPX 286 Processor Version 1 is also referred to as "UNIX System V/286" in this and other UNIX system documents.

List of Examples

- Example 1. System Boot
- Example 2. Loading Disk Parameters with **fdisk**
- Example 3. Choosing to Display Partitions
- Example 4. Returning to **fdisk** Operations Menu
- Example 5. Choosing to Create a Partition
- Example 6. Creating a UNIX Partition
- Example 7. Creating a DOS Partition
- Example 8. Returning to **fdisk** Operations Menu
- Example 9. Making the UNIX Partition Active
- Example 10. Returning to **fdisk** Operations Menu
- Example 11. Scanning for Bad Tracks
- Example 12. Leaving **fdisk**
- Example 13. Viewing Default Allocation with **divvy**
- Example 14. Updating the Hard Disk with Changes
- Example 15. Down-Loading Files
- Example 16. Installing **runtime.1**
- Example 17. Installing **runtime.2**
- Example 18. Logging in as **root**
- Example 19. Installing **exruntime.1**

STEP 1

Overview and Introduction

Installation of the complete system should take one to two hours depending upon the size of your hard disk.

Note that a minimum of 512k bytes of RAM memory is required for the installation of System V/AT.

There are two installation procedures available on the System V/AT boot floppy: *easyinstall* and *installit*.

Both *easyinstall* and *installit* are fairly easy to use. If you understand how to partition the hard disk by cylinder number ranges, similar to the DOS FDISK utility you probably want to use *installit*. Otherwise, use *easyinstall*. UNLESS you wish to do any of the following:

1. You wish to change the default allocations for the root and /usr file systems, or the system swap area. The defaults are as follows:

12000 blocks (6 megabytes)	— for root
4000 blocks (2 megabytes)	— for the system swap area
the remainder of the disk	— for /usr
2. You wish to install System V/AT on the secondary as well as the primary hard disk, or
3. You wish to preserve a System V partition on your hard disk.

To use *easyinstall*, follow the installation procedures through Step 2 of this chapter. At this point, you will be instructed to type *easyinstall* instead of continuing through the Steps.

INSTALLATION INSTRUCTIONS

Installing System V/AT is a fairly straight-forward operation, but we recommend that you first read the entire procedure.

The basic installation consists of the following components:

- 1) Inserting your System V/AT boot disk into your floppy drive;
- 2) Rebooting your system;
- 3) Formatting of your hard disk (ONLY if needed) with **format**;
- 4) Configuring your hardware with **setup**; and
- 5) Actually installing System V/AT with **installit** which:
 - Creates your disk partitions and bad track table with **fdisk**,
 - Allocates disk space for your UNIX file system with **divvy** and **mkfs**, and
 - Copies System V/AT files to your hard disk.

If you do not know the specifications of your hard disk drive(s), check the drive's owner's manual.

Instructions are provided for systems with different configurations.

Example consoles can be found after the portion of the installation procedure they are intended to illustrate. They represent a typical installation of System V/AT on an IBM PC-AT or an AT-compatible with a single 20-megabyte formatted hard disk. Roughly 17 megabytes will be used for an active UNIX partition and the remainder for a DOS partition.

Before you begin this installation procedure, it is advisable to be backed up on your hard disk files. Use a backup procedure such as the **cpio** command (as documented in Section 1 of the Runtime System manual), to copy to floppy disks whichever files you wish to save from your hard disk. You can then reload them after the installation is completed.

STEP 2

Booting System V/AT

Instructions

C-2

If you have previously installed System V/AT and now only wish to upgrade your UNIX kernel, see Appendix #1 of this chapter.

If possible, back up your boot floppy before attempting to boot up System V/AT. You may do this by using the DOS **diskcopy** utility as described in your DOS documentation. Always use a double sided double density (1.2 MB) diskette for the target media. Depending on your DOS version, the copy you make using the **diskcopy** utility may not perform satisfactorily as System V/AT media. You can also make a backup copy of your boot floppy under UNIX after you are installed.

Remove the disk labeled "bootdisk.1" from its jacket at the beginning of the Section 1 of the Runtime System manual and insert it into your floppy disk drive.

This must be a 1.2 megabyte drive.

You should NOT place a write-protect tab (thereby covering the rectangular notch) on the boot floppy.

Reset the machine by pressing the Ctrl, Alt and Del keys simultaneously.

The System V/AT banner should appear on your screen within two minutes.

If, after you first get the "#" prompt, your screen flickers, type "unflicker" to configure System V/AT for your particular display hardware.

IMPORTANT NOTE: When you see the "#" prompt, your system is rooted in the boot floppy file system. System V/AT treats this file system just like a hard disk rooted file system, and you should never remove the boot floppy from the drive without properly shutting down.

INSTALLATION INSTRUCTIONS

At the end of the boot floppy installation procedure, you will be properly shut down and you will be instructed to remove the floppy and reboot. If you wish to shut down and remove the boot floppy at any other time, you must issue the following command, which will also initiate the re-boot procedure for your system:

```
PROMPT      #  
RESPONSE    sync; sync; sync; init 0
```

This is the **ONLY** recommended way to remove the boot floppy before the end of the installation procedure.

Using setup

When you get the # prompt, type **setup** to see a list of options that allows you to configure your hardware, as follows:

```
PROMPT      #  
RESPONSE    setup
```

Use **setup** to display or configure your hard disk drive type as follows:

Type **setup fixed 1** to display the **primary** hard disk drive type currently stored in **cmos** on your system.

Type **setup fixed 1 [1-99]** to change the value stored for the primary hard disk drive type to the type recommended by the manufacturer of your drive.

If your hard disk drive is a non-standard drive and the drive's manufacturer does not specify a drive type number to use, use a type which specifies the same number of heads as your drive. A table of some manufacturer's drive parameters, and the values for the pre-stored ("standard") types are found in Appendix 5 to this chapter.

If your hard disk drive is a non-standard drive and the drive's manufacturer specifies a drive type number to use, use that number.

```
PROMPT      #
RESPONSE    setup fixed 1 #
```

where the final number (#) is the number you have chosen for your hard disk drive type number.

```
PROMPT      Are you sure? (type y or n)
RESPONSE    y
```

If you are uncertain about the present condition of your bad track table, type **showbad** to display it, as follows:

```
PROMPT      #
RESPONSE    showbad
```

If you also plan to install System V/AT on a secondary hard disk, see Appendix #2 of this chapter.

Use other **setup** options as needed to configure your hardware.

Using easyinstall

If you are using **easyinstall** (see Step 1), proceed as follows:

```
PROMPT      #
RESPONSE    easyinstall
```

The **easyinstall** procedure will interactively guide you through the installation.

Otherwise, continue the installation using the following instructions.

Formatting the hard disk

Instructions

You may now format your (primary) hard disk.

Formatting of a hard disk is usually done by the disk manufacturer before the disk is sold; therefore, only format your hard disk if you know that you need to do so. If you are in doubt as to whether you need to format your hard disk, then DO NOT do so.

If you have any files on your hard disk that you wish to save, be sure to do so before formatting it. Formatting destroys ALL files currently on your hard disk.

Formatting your hard disk is done with System V/AT's `format` command. For your primary hard disk use the following syntax:

```
/etc/format -s /dev/rdisk/0s10
```

If you get errors using the above syntax, you may need to use the following procedure. First format your drive without using the `-s` option:

```
/etc/format /dev/rdisk/0s10
```

Then run the `fdisk` utility with the `-s` option to copy the boot block onto the hard disk, after formatting. The syntax is:

```
PROMPT      #
RESPONSE    fdisk -s
```

After running `fdisk`, and changing partition data, so that `fdisk` writes its changes to the hard disk, you can go on to the next step, the `installit` command.

STEP 3**Beginning Installation of System V/AT****Instructions****C-2**

The **installit** command first invokes the **fdisk** utility and then later the **divvy** utility, in order to set up the hard disk file system.

To return to the operating system level (i.e., to get back the "#" prompt), press the **Del** key.

To begin the installation, you are prompted with:

1. *PROMPT* #
RESPONSE **installit**
2. *PROMPT* Are the hard disk partitions set up correctly?
 (y or n):

Type **y** ONLY IF the following four conditions are ALL true:

- a) you have already established a bad track table under the **fdisk** utility,
- b) you have not changed the disk partitions since the time at which that bad track table was established,
- c) you do not wish to now change either the disk partitions or the bad track table, and
- d) you do not wish to partition a secondary hard disk.

RESPONSE **y**

If you typed **y**, skip to Step 8 (*Beginning the Use of the divvy Utility - Viewing Default Allocation of UNIX Partition*) of this chapter.

Otherwise, to invoke **fdisk**:

ALTERNATE
RESPONSE **n**

Example

Example Console 1
System Boot

```
Microport's System V/AT Release 2 AT Version 1.3.6
Copyright (c) 1985 AT & T           - All Rights Reserved
Copyright (c) 1985 Microport       - All Rights Reserved

INIT: SINGLE USER MODE

To install Microport's System V/AT, type the following
commands:

unflicker   - if your console screen flickers
setup       - to configure your hardware
installit   - to set up and install your System V software

# setup fixed 1 2
Set first fixed disk drive as type 2
Are you sure? (type y or n) y
Cmos changed
# installit
Are the hard disk partitions set up correctly? (y or n) : n
```

Notes

The above assumes that your primary hard disk is a standard type-2 hard disk drive or has the same number of heads as a standard type-2 drive.

STEP 4

Beginning the use of the fdisk Utility — Loading Parameters of Your Hard Disk

C-2

Instructions

The **fdisk** utility allows you to modify the hard disk partitions and build a table of bad tracks.

If your hard disk drive is a standard type,

when **fdisk** comes up, it displays that drive type on your screen.

<i>PROMPT</i>	Is this correct:
<i>RESPONSE</i>	y

and continue the installation procedure with Step 5 (*Displaying Current Configuration of Partitions for Your Hard Disk*) of this chapter.

INSTALLATION INSTRUCTIONS

Example

Example Console 2 Loading Disk Parameters with fdisk

Executing /etc/fdisk, to establish the hard disk partitions
(one moment please)

drive 0 type= 2

NO PARTITION TABLE

drive 1 type= 0

Unit 0 Drive Type in CMOS RAM is 2

This will cause the system to assume the following drive
parameters:

Cylinders	Tracks/Cylinder	Landing Zone	Write Precomp
615	4	615	300

All drives must be formatted with 17 512-byte sectors per track.

Is this correct: **y**

Notes

The above instructions assume that your primary hard disk drive is a standard type-2 drive.

Using a Non-standard type disk drive

Instructions

If you have a non-standard-type hard disk drive: when **fdisk** comes up, it should display that drive type on your screen and ask you:

- | | | |
|----|-----------------|---|
| 1. | <i>PROMPT</i> | Is this correct: |
| | <i>RESPONSE</i> | n |
| 2. | <i>PROMPT</i> | enter drive type: |
| | <i>RESPONSE</i> | 0 |
| 3. | <i>PROMPT</i> | enter number of cylinders: |
| | <i>RESPONSE</i> | type the correct number of cylinders for your drive |
| 4. | <i>PROMPT</i> | enter number of heads per cylinder: |
| | <i>RESPONSE</i> | type the correct number of heads for your drive |
| 5. | <i>PROMPT</i> | type the correct landing zone cylinder number for your drive |
| | <i>RESPONSE</i> | |
| 6. | <i>PROMPT</i> | enter cylinder no. for write precompensation: |
| | <i>RESPONSE</i> | type the correct write precompensation cylinder number for you drive (if your drive has no write precompensation, type -1 ; if in doubt about the correct number, use the corresponding value of the comparable IBM drive type) |

When the new drive parameters are displayed:

RESPONSE **y**

Notes

The terms *heads* and *tracks* are used interchangeably.

STEP 5

Displaying Current Configuration of Partitions for Your Hard Disk

Instructions

The **fdisk** utility displays a partition's size in 512-byte blocks. The number of blocks per cylinder is found by multiplying the number of tracks per cylinder by 17.

fdisk requires one active UNIX partition on your primary hard disk. The minimum partition size recommended for the installation of the complete system is 37,020 blocks for your primary hard disk. If you also wish to create a UNIX file system on a secondary hard disk, you must create an active UNIX partition on that disk and set aside an additional 1,020 blocks for the overhead tracks for that disk. (For a given partition on a hard disk, the actual number of blocks available for the UNIX file systems and the system swap area is equal to the partition size in blocks minus the 1,020 blocks used for overhead tracks. So, in order to provide UNIX with 36,000 blocks, a partition of 37,020 blocks must be created.)

The number and size of the active partitions you need to create depends upon the number and size of the different systems you wish to simultaneously reside on your hard disk. If, at some point in the future, you decide you need to change the partitioning of your hard disk, you can do that by repeating this installation procedure. The type of your drive will determine the number of cylinders that must be allocated to create a partition of a desired size.

fdisk displays the following menu (see Example Console 3):

<i>PROMPT</i>	Enter choice <cr>:....:
<i>RESPONSE</i>	4

Example

Example Console 3
 Choosing to Display Partitions

C-2

```

Invalid Partition Table ! Clearing !
-----
Microport's Fixed Disk Setup Program

FDISK Options:

Choose one of the following:
  1. Create a Partition
  2. Change Active Partition
  3. Delete a Partition
  4. Display Partition Information
  5. Scan and Assign Bad Tracks
  6. Advance to next disk unit-Current unit is [ 0 ]

Press 'x' to return to UNIX
Enter Choice <cr>:.....: 4
    
```

Instructions

Choose **4** to view the Displaying Partition Information Menu
 (Example Console 4)

<i>PROMPT</i>	Press 'x' to return to FDISK options
<i>RESPONSE</i>	x

NOTE: In the above display, menu option "Advance to next disk unit" refers to installing System V/AT on a secondary (as well as a primary) hard disk. To partition your secondary hard disk, see Appendix #2.

INSTALLATION INSTRUCTIONS

Example

Example Console 4
Returning to **fdisk** Options Menu

Microport's Fixed Disk Setup Program

Display Partition Information

Partition	Status	Type	Start	End	Size	Blocks
4	N	unknown	0	0	0	0
3	N	unknown	0	0	0	0
2	N	unknown	0	0	0	0
1	N	unknown	0	0	0	0

Press 'x' to return to FDISK options: X

Notes

The above screen will be displayed if your hard disk has no partitions. If there is already a DOS partition on your hard disk it will be displayed at this time.

STEP 6

Partitioning Your Hard Disk

Instructions

C-2

If your hard disk already has a partition table (there is at least one non-zero integer in the blocks column of the partition table display) but you wish to change it, each partition you wish to change must first be deleted. Otherwise continue the installation on the following page, "Creating a UNIX partition on your hard disk."

1. *PROMPT* Enter choice <cr>:....:
RESPONSE 3 (from the **fdisk** menu)

2. *PROMPT* Which partition do you wish to delete?
RESPONSE Type the appropriate integer between
4 and 1. (Note the reversed order in which
the partitions are displayed.)

3. *PROMPT* Do you want to continue...?:

If you do not wish to delete another partition,

RESPONSE n

Creating a UNIX partition on your hard disk.

PROMPT Enter choice <cr>: ...:
RESPONSE 1 (from the **fdisk** menu)

Example

Example Console 5
Choosing to Create a Partition

```
Microport's Fixed Disk Setup Program
FDISK Options:

Choose one of the following:
  1. Create a Partition
  2. Change Active Partiton
  3. Delete a Partition
  4. Display Partition Information
  5. Scan and Assign Bad Tracks
  6. Advance to next disk unit-Current unit is [ 0 ]

Press 'x' to return to UNIX
Enter choice <cr>:.....: 1
```


Instructions

- | | |
|--|--|
| <p>1. <i>PROMPT</i></p> <p style="padding-left: 20px;"><i>RESPONSE</i></p> | <p>Which partition would you like to create [1-4]?...:
 You may type any single integer from 1 to 4. (Note the reversed order in which the partitions are displayed. It makes no difference which partition number you use.)</p> |
| <p>2. <i>PROMPT</i></p> <p style="padding-left: 20px;"><i>RESPONSE</i></p> | <p>Enter Starting Cylinder Value:
 Type the appropriate positive integer. (Note that a partition may not begin at cylinder 0.)</p> |
| <p>3. <i>PROMPT</i></p> <p style="padding-left: 20px;"><i>RESPONSE</i></p> | <p>Enter Ending Cylinder Value:
 Type the appropriate positive integer (less than the total number of cylinders available on your drive).</p> |
| <p>4. <i>PROMPT</i></p> <p style="padding-left: 20px;"><i>RESPONSE</i></p> | <p>What operating system (1=DOS, 5=UNIX)?:
 5</p> |

C-2

Example

Example Console 6
Creating a UNIX Partition

```
Microport's Fixed Disk Setup Program

Create a Partition

Partition  Status  Type  Start  End  Size  Blocks
   4         N   unknown  0     0    0     0
   3         N   unknown  0     0    0     0
   2         N   unknown  0     0    0     0
   1         N   unknown  0     0    0     0

Press 'x' to return to FDISK options

Which partition would you like to create [1-4]? ...: 4
Enter Starting Cylinder Value: 1
Enter Ending Cylinder Value: 510
What operating system (1=DOS, 5=UNIX)? : 5
```

Notes

You may choose to create a UNIX partition in any of the four partitions. Partition 4 has been arbitrarily chosen for this example. The above will result in a 17-megabyte (approximately) UNIX partition. If a DOS partition already exists, and the space it occupies allows the proper sized System V/AT to be created, you may leave the DOS partition on your drive. If you do leave a DOS partition on your drive, you should make sure your DOS data is backed up, just in case of problems with the installation.

Creating a DOS Partition

Instructions

Creating a DOS partition on your hard disk is optional. The number and size of the active partitions you need to create depends upon the number and size of the different systems you wish to simultaneously reside on your hard disk. If, at some point in the future, you decide you need to change the partitioning of your hard disk, simply repeat this installation procedure.

If you wish to create a DOS partition in any of the remaining partitions, repeat the above procedure except:

```
PROMPT          What operating system (1=DOS, 5=UNIX)?
RESPONSE        1
```

Example

Example Console 7 Creating a DOS Partition

Microport's Fixed Disk Setup Program

Create a Partition

Partition	Status	Type	Start	End	Size	Blocks
4	N	System5	1	510	510	34680
3	N	unknown	0	0	0	0
2	N	unknown	0	0	0	0
1	N	unknown	0	0	0	0

Press 'x' to return to FDISK options

Which partition would you like to create [1-4]?...: **3**

Enter Starting Cylinder Value: **511**

Enter Ending Cylinder Value: **614**

What operating system (1=DOS, 5=UNIX)?: **1**

INSTALLATION INSTRUCTIONS

Notes

The cylinder values of the partitions may NOT overlap.

Creating a DOS partition is optional, and you can do so in any one of the remaining partitions. Partition 3 has been arbitrarily chosen for this example. The above will result in a 3.5-megabyte DOS partition.

The maximum cylinder number used must be equal to 1 less than the total number of cylinders on the drive, which are numbered from zero.

If a DOS partition already exists, and the space it occupies allows the proper sized System V/AT to be created, you may leave the DOS partition on your drive. If you do leave a DOS partition on your drive, you should make sure your DOS data is backed up, just in case of problems with the installation.

To return to the **fdisk** options menu:

RESPONSE **x**

Example

Example Console 8 Returning to **fdisk** Options Menu

Microport's Fixed Disk Setup Program

Create a Partition

Partition	Status	Type	Start	End	Size	Blocks
4	N	System5	1	510	510	34680
3	N	DOS	511	614	104	7072
2	N	unknown	0	0	0	0
1	N	unknown	0	0	0	0

Press 'x' to return to FDISK options

Which partition would you like to create [1-4]?...: **x**

Activating the UNIX partition

1. *PROMPT* Enter choice <cr>:....:
RESPONSE 2 (from the **fdisk** menu)
2. *PROMPT* Enter the number of the partition you want
to make active (1-4)...:
RESPONSE Type the **number of the UNIX partition**. (Note the reversed order in which the partitions are displayed.)

Example

Example Console 9
Making the UNIX Partition Active

```

Press 'x' to return to UNIX
Enter choice <cr>:.....: 2
Microport's Fixed Disk Setup Program

Change the Active Partition

Partition  Status  Type      Start  End  Size  Blocks
   4         N   System5    1    510  510  34680
   3         N     DOS      511   614  104   7072
   2         N   unknown    0     0    0     0
   1         N   unknown    0     0    0     0

Type 'x' to return to FDISK Options
Enter the number of the partition you want to make active
(1-4) .....: 4
    
```

Notes

The operating system that resides in the active partition will be used each time you boot your computer.

If you are also installing System V/AT on a secondary disk drive, that drive will also require one active partition.

INSTALLATION INSTRUCTIONS

Again, you will be asked:

PROMPT Enter the number of the partition you want
 to make active (1-4)...:
RESPONSE **x**

This returns you to the **fdisk** menu.

Example

Example Console 10
Returning to fdisk Options Menu

```
Microport's Fixed Disk Setup Program

Change the Active Partition

Partition    Status    Type        Start    End    Size    Blocks
  4           A       System5      1      510    510    34680
  3           N       DOS           511    614    104    7072
  2           N       unknown      0      0      0      0
  1           N       unknown      0      0      0      0

Type 'x' to return to FDISK Options

Enter the number of the partition you want to make active 1-4).....: x
```

STEP 7**Creating a Bad Track Table for Your Hard Disk****Instructions**

The bad track table is a listing of bad tracks on your hard disk which also specifies alternate tracks to be used instead. If you have not established a bad track table under the **fdisk** utility since the last time you changed your disk partitions, but have an active UNIX partition on your hard disk, you must now create a new bad track table. This may be done either automatically but slowly (10-90 minutes) by having the system scan the entire hard disk for bad tracks, or else done quickly, if the data is already known, by entering it manually. The manufacturer's list of known bad tracks is usually found attached to the physical drive inside your machine.

To automatically build a bad track table, scan for bad tracks (bad areas on your hard disk).

- | | | |
|----|-----------------|--|
| 1. | <i>PROMPT</i> | Enter choice <cr>:....: |
| | <i>RESPONSE</i> | 5 |
| 2. | <i>PROMPT</i> | Is a complete bad track table to be written: |
| | <i>RESPONSE</i> | y |
| 3. | <i>PROMPT</i> | Do you wish to scan the disk for bad tracks: |

If you already know the location of each bad track on your hard disk drive and wish to manually enter this information:

RESPONSE n

Otherwise:

ALTERNATE
RESPONSE y

If you choose y, the scanning process will now take place (anywhere from 20 to 90 minutes will be required for this, depending upon the size of your hard disk drive). You may wish to write down the location of each of the bad tracks discovered by the scan. Thus, if you ever need to rebuild your bad track table, you can do so more quickly by manually entering the necessary information.

Example

Example Console 11
Scanning for Bad Tracks

FDISK Options:

Choose one of the following:

1. Create a Partition
2. Change Active Partition
3. Delete a Partition
4. Display Partition Information
5. Scan and Assign Bad Tracks
6. Advance to next disk unit-Current unit is [0]

Press 'x' to return to UNIX

Enter choice <cr>:.....: **5**

Is a complete new bad track table to be written: **y**

Do you wish to scan the disk for bad tracks: **y**

Instructions

You can now manually build your bad track table, or enter any bad tracks which appear on the bad track list supplied with your drive, but were not found by **fdisk**. If you don't have such a list, assume for now that the scan was able to discover all the bad tracks. Note that you must terminate the process of entering additional bad tracks by pressing **Ctrl-d**.

1. **PROMPT** Do you wish to type in bad tracks:

If you have no additional bad track information:

RESPONSE n

and continue the installation procedure at the next step of this chapter.

Otherwise:

ALTERNATE
RESPONSE y

2. **PROMPT** Enter bad track as: cylinder, track:

If you have an additional bad track to enter:

RESPONSE Type the cylinder number followed
by a comma and its track number.

For example, to enter cylinder 128, track 2 into the bad track list, type

RESPONSE 128,2

When you have no additional bad track information:

3. **PROMPT** Enter bad track as: cylinder, track:
RESPONSE ctrl-d

This returns you to the **fdisk** options menu.

Leaving fdisk

If you do NOT wish to partition an additional hard disk, you may now exit from the **fdisk** utility.

1. *PROMPT* Enter choice <cr>:....:
RESPONSE **x**

2. *PROMPT* Leaving FDISK - Do you want your changes
 installed on the previous unit? (y or n)?
 (Note that "previous unit" here refers to the
 hard disk for which you have just created
 a partition.)

RESPONSE **y**

3. *PROMPT* Are the hard disk partitions set up correctly
 NOW? (y or n):

RESPONSE **y**

Example

Example Console 12 Leaving fdisk

C-2

Scancyl614trk3

Do you wish to type in bad tracks: **n**

Bad Track Update Complete

Microport's Fixed Disk Setup Program

FDISK Options:

Choose one of the following:

1. Create a Partition
2. Change Active Partition
3. Delete a Partition
4. Display Partition Information
5. Scan and Assign Bad Tracks
6. Advance to next disk unit-Current unit is [0]

Press 'x' to return to UNIX

Enter choice <cr>:.....: **x**

Leaving FDISK - Do you want your changes installed on the previous unit?

- (y or n)? **y**

Updating disk unit 0 the hard disk will now be reinitialized

Are the hard disk partitions set up correctly NOW? (y or n): **y**

Notes

The above assumes that there is not additional bad track information available.

If you wish to partition an additional hard disk, see Appendix #2 at the end of this chapter.

STEP 8**Beginning the Use of the divvy Utility —
Viewing Default Allocation of UNIX Partition****Instructions**

It is assumed throughout the following that you wish to establish new file systems for the active System V partition. You must re-establish new file systems whenever you change the boundaries of the System V partition, or create a new bad track table. If you do not wish to establish or re-establish new file systems, proceed to Step 10.

<i>PROMPT</i>	Have you set up the file systems correctly? (y or n):
<i>RESPONSE</i>	n

divvy will be invoked, and default disk space allocations will be displayed for the `/root` and `/usr` file systems, and for the system swap area (note that any zero size allocation, such as the default allocation for the `/tmp` file system, does not show up on your screen).

Note that normally if you are now told, **NO PARTITION TABLE**, this will refer to your secondary hard disk.

<i>PROMPT</i>	Do you wish to change any allocation? (y or n):
---------------	--

If you wish to use the default allocations:

<i>RESPONSE</i>	n
-----------------	---

For information to consider if you do not wish to use the default disk space allocations, see Appendix #3 of this chapter.

Example**Example Console 13**
Viewing Default Allocation with **divvy**

Have you set up the file systems correctly: (y or n): **n**
Executing **/etc/divvy**, to establish the filesystems (one moment please)
drive 0 type= 2
drive 1 type= 0

DEVICE	UNIT	BLOCKS
/root	0	12000
/swap	0	4000
/usr	0	17680

Do you wish to change any allocation? (y or n): **n**

Notes

The above creates a 6-megabyte **/root** file system, a 2-megabyte system swap area, and an 8.5-megabyte **/usr** file system (roughly) in the UNIX partition.

STEP 9

Continuing the Use of the divvy Utility — Making the File Systems

Instructions

divvy will now invoke the **mkfs** utility to make the UNIX file systems:

1. *PROMPT* Shall I update the disk? (y or n):
 RESPONSE y

2. *PROMPT* Shall I proceed? (y or n):
 RESPONSE y

3. *PROMPT* Shall I proceed? (y or n):
 RESPONSE y

Example

Example Console 14 Updating the Hard Disk with Changes

Ready to write unit #0-
Shall I update the disk? (y or n): **y**
the hard disk will now be reinitialized

Ready to make the following file system:
mkfs /dev/dsk/0s0 12000:1200 2 68

Shall I proceed? (y or n): **y**
drive 0 type= 2
drive 1 type= 0
Mkfs: /dev/dsk/0s0?
(DEL if wrong)
bytes per logical block = 1024
...
...
cylinder size (physical blocks) = 68

Ready to make the following file system:
mkfs /dev/dsk/0s2 17660:1766 2 68

Shall I proceed? (y or n): **y**

Notes

Once the process of making the file system has actually begun, if an error is detected, you can stop the process by striking the Del key.

The "..." in the above example indicates a portion of the actual console display that has been deleted for the sake of brevity.

STEP 10

Down-Loading Files from the Boot Disk to Your Hard Disk

Instructions

PROMPT Have you set up the file systems correctly
 NOW? (y or n):
RESPONSE y

Example

Example Console 15
Down-Loading Files

```
gap (physical blocks) = 2
cylinder size (physical blocks) = 68
Have you set up the file systems correctly NOW? (y or n): y

The hard disk is now configured for system5
Copying over the necessary files:
+ label it /dev/rdisk0s0 root 0s0
Currentfsname: , Current volname: , Blocks: 12000,
  Inodes: 1200
FS Units: 1Kb, Date last mounted: Mon Jul 7 20:14:13 1986
...
...
...
+ set +x
The hard disk is now initialized.
Please wait for the message 'The system can now be rebooted'.
After the system has come to a stop, remove the floppy in
  drive A, and reboot your computer (via control-alt-del).
When you see the # prompt, type 'installit' to finish the
  installation.

INIT: New run level: 0
The system can now be rebooted
```


Notes

The y response begins the process of down-loading the files from the boot disk. Once the process of down-loading the files from the boot disk has actually begun, if an error is detected, down-loading can be stopped by striking the Del-key. Note that as each file is down-loaded, its path name will appear on your screen.

The "... " indicates a portion of the actual console display that has been deleted for the sake of brevity.

When the console message appears "The system can now be rebooted" it is safe to remove the boot floppy from the drive and re-set your system.

STEP 11

Installing Runtime System

Instructions

When you see the # prompt, you may continue with the installation.

- PROMPT* #
ACTION Insert the floppy disk labeled **runtime.1**
RESPONSE **installit**
- PROMPT* Is this ok? (y or n)
RESPONSE **y**

Example

Example Console 16
Installing "runtime.1"

Microport's System V/AT Release 2 AT Version 2.2
Copyright (c) 1985 AT&T - All Rights Reserved
Copyright (c) 1985 Microport - All Rights Reserved

INIT: SINGLE USER MODE

To continue installing Microport's System V/AT, insert the floppy
labeled 'runtime.1' and type 'installit'

installit

1 blocks

Installing from #1 of set runtime

Copyright (c) 1985 AT&T
All Rights Reserve

Is this OK? (y or n) **y**

Instructions

<i>PROMPT</i>	Insert floppy set runtime #2 then type installit
<i>ACTION</i>	Remove the floppy disk that is currently in drive A.
<i>ACTION</i>	Insert the floppy disk labeled runtime.2
<i>RESPONSE</i>	installit

Example

Example Console 17
Installing "runtime.2"

```
usr/adm/sa
usr/bin
usr/bin/at
usr/bin/banner
usr/bin/batch
usr/bin/bfs
usr/bin/cal
usr/bin/calendar
usr/bin/comm
usr/bin/cpset
usr/bin/crontab
usr/bin/ct
usr/bin/cu
usr/bin/getopt
usr/bin/id
usr/bin/logname
usr/bin/man
1915 blocks

insert floppy set runtime #2 then type 'installit'
# installit
```

INSTALLATION INSTRUCTIONS

After the installation of **runtime** is complete, you will be logged out as the system goes into multi-user mode.

Remove the floppy disk from drive A.

Example

Example Console 18
Logging in as "root"

```
+ chmod 775 /usr
+ mv /etc/release/.profile /
+ mv /etc/release/motd /etc
+ chown root /.profile
+ chgrp sys /.profile
+ chmod 444 /.profile
+ chown root /etc/motd
+ chgrp sys /etc/motd
+ chmod 644 /etc/motd
+ /bin/rm -r /etc/release
+ chmod 444 /etc/install.date
+ set +x
runtime set loaded, going multi-user. You may load any package
by inserting the floppy and typing installit.
INIT: New run level: 2
cron not started
System name: system5
Console Login: root
```

Instructions

You may now log on as **root** (super-user) and install the extra runtime disk, the "utility" disk, and any of the other System V/AT packages you wish (assuming you have sufficient space on your hard disk). If you wish to list the contents of any of these disks, insert that disk into drive A and then type

```
cpio -it < /dev/rdisk/0s25.
```

Specific installation instructions appear on succeeding pages.

If you wish to install the extra runtime disk, or the "utility" disk

- | | | |
|----|-----------------|---|
| 1. | <i>PROMPT</i> | Insert the floppy disk labeled "exruntime.1" or "utility.1" |
| | <i>RESPONSE</i> | installit |
| 2. | <i>PROMPT</i> | Is this ok? (y or n) |
| | <i>RESPONSE</i> | y |

After the installation of the **exruntime.1** disk, or the "utility" disk is complete, remove the floppy disk from drive A.

Example

Example Console 19
Installing "exruntime.1"

Microport's System V/AT
System V Release 2 for PC-AT compatible systems
including
File Hardening & Record Locking Features
Release 2.2
Please report to Microport Systems, Inc.
10 Victor Square, Scotts Valley, CA 95066
(408) 438-8649

news: releasenotes
installit
1 blocks
Installing from #1 of set exruntime
Copyright (c) 1985 AT&T
All Rights Reserved
Is this ok? (y or n) **y**

Notes

If you are also installing System V/AT on a secondary hard disk, see Appendix #2 of this chapter.

STEP 12**Installing Software Development System****Instructions**

If you are not already logged in as root, do so now.

1. *PROMPT* #
ACTION Insert the floppy disk labeled progdev.1.
RESPONSE **installit**
2. *PROMPT* Is this ok? (y or n)
RESPONSE y
3. *PROMPT* Insert floppy set progdev #2 then type
installit
ACTION Remove the floppy disk currently in
drive A.
ACTION Insert the floppy disk labeled progdev.2.
RESPONSE **installit**
4. *PROMPT* Insert floppy set progdev #3 then type
installit
ACTION Remove the floppy disk currently in
drive A.
ACTION Insert the floppy disk labeled progdev.3.
RESPONSE **installit**

After the installation of **progdev** is complete, remove the floppy disk from drive A.

To also install the extra **progdev** disk:

1. *ACTION* Insert the floppy disk labeled **exprogdev.1**
RESPONSE **installit**
2. *PROMPT* Is this ok? (y or n)
RESPONSE y
3. *ACTION* Remove the floppy disk from drive A.

If you wish to install an upgrade disk, which updates your runtime and software development utilities to a new version level, you should do so at this time, by typing *installit*.

Step 13

Installing Text Preparation System

Instructions

C-2

If you are not already logged in as **root**, do so.

1. *ACTION* Insert the floppy disk labeled **text.1**
RESPONSE **installit**
2. *PROMPT* Is this ok? (y or n)
RESPONSE **y**

After the installation of **text** is complete, remove the floppy disk from drive **A**.

To also install the extra text disks:

1. *ACTION* Insert the floppy disk labeled **extext.1**
RESPONSE **installit**
2. *PROMPT* Is this ok? (y or n)
RESPONSE **y**
3. *PROMPT* Insert floppy set **extext #2** then type
installit
ACTION Remove the floppy disk currently in
drive **A**.
ACTION Insert the floppy disk labeled **extext.2**.
RESPONSE **installit**
ACTION Remove the floppy disk from drive **A**.

STEP 14

Ready to Go

The hard disk installation of System V/AT is now completed. Reboot and log in as **root**. At this point no password is needed for the root login, but, in order to avoid unauthorized use of your system, one should be assigned by using the **passwd** command as documented in Section 1 of the Runtime System manual.

Additional System Options

To add new **login names** to the system, see Appendix # 4 of this chapter.

To install a **modem**, see **sio (7)** in the Runtime System manual.

Appendix 1

Upgrading UNIX Kernel

Instructions

The kernel is a file called `/system5` on your system; the filename `/unix` is symbolically linked to the filename `/system5`.

- 1.1 To upgrade your UNIX kernel, and appropriate utilities from the boot floppy

Boot your system from the boot floppy, and type *installit*; answer y to the question "do you wish to upgrade an existing System V/AT installation"

```

PROMPT      #
RESPONSE    installit

PROMPT      upgrade an existing System V/AT
RESPONSE    y
    
```

When you see the message IN 10 "The system can now be rebooted" you can safely remove the boot floppy from the drive, and reset your system. To continue with the upgrade, follow the Release Notes to install any additional floppies, usually by inserting the floppy in the drive and typing *installit*.

- 1.2 An alternative method of upgrading only your UNIX kernel to the "small" kernel version on the boot floppy is given below. Note that this is not the normal method of upgrading your UNIX kernel.

1. ACTION Mount the boot floppy on /mnt:
RESPONSE mount /dev/dsk/0s25 /mnt
2. ACTION Copy the new kernel onto the old one:
RESPONSE cp /mnt/system5 /
3. ACTION Patch the new kernel to boot properly:
RESPONSE etc/hdrt.patch /system5
4. ACTION Unmount the boot floppy:
RESPONSE nmount /dev/dsk/0s25

Shutdown and reboot from your hard disk.

Appendix 2

Installing System V/AT on a Secondary Hard Disk

Instructions

- 2.1 You may need to format your secondary hard disk.

Formatting of a hard disk is usually done by the disk manufacturer before the disk is sold; therefore, only format your hard disk if you know that you need to do so.

Formatting your hard disk is done with System V/AT's **format** command. For your secondary hard disk use the following command. For your secondary hard disk use the following syntax:

```
/etc/format /dev/rdisk/1s10
```

For further information, check Section 1 of the Runtime System manual.

- 2.2 Run **setup** for your secondary hard disk.

If your secondary hard disk drive is a standard drive, type:

```
setup fixed 2 [1-99]
```

where the final number is your drive type number.

If your secondary hard disk drive is a non-standard drive, type:

```
setup fixed 2 [1-99]
```

where the final number is either the manufacturer's suggested drive type number (if one is available) or else a drive number you have selected for your drive. See Appendix 5.

```
PROMPT      Are you sure? (type y or n)
RESPONSE   y
```

The installation procedure may be continued with Step 3, (*Beginning Installation of System V/AT*), of this chapter.

- 2.3 Create an active UNIX partition on your secondary hard disk (unit 1).
- | | | |
|----|-----------------|--|
| 1. | <i>PROMPT</i> | Enter choice <cr>:....: |
| | <i>RESPONSE</i> | 6 |
| 2. | <i>PROMPT</i> | Do you want your changes installed on the previous unit? (y or n)? |
| | <i>RESPONSE</i> | y |

Note that "previous unit" here refers to the hard disk for which you have just created a partition.

For your secondary hard disk, execute the following set of procedures as described earlier in this chapter:

Step 4: Beginning the Use of the fdisk Utility—Loading Parameters of Your Hard Disk (note that after you are told "Updating disk unit 0", all subsequent prompts requesting disk partitioning parameters will be referring to your secondary disk),

Step 5: Displaying Current Configuration of Partitions for Your Hard Disk,

Step 6: Creating an Active UNIX Partition on Your Hard Disk, and

Step 7: Creating a Bad Track Table for Your Hard Disk.

You can continue the installation procedure at Step 8 (*Beginning the Use of the divvy Utility—Viewing Default Allocation of UNIX Partition*) of this chapter.

Notes

To create and mount the file systems on your secondary hard disk, use the **mkfs** and **mount** commands, respectively. For further information, see the Runtime System manual.

Appendix 3

Changing Allocation of UNIX Partition

The UNIX partition must now be divided between the `/root` file system, the `/usr` file system, and the system swapping area. The total number of blocks that the `divvy` command allocates is equal to the number of blocks in the UNIX partition minus 1,020 blocks for overhead tracks. Of this amount, you should allocate at least 12,000 blocks for `/root` and 4,000 blocks for `/swap`. Increasing the system swap area beyond 4,000 blocks will have no effect unless you are running several users.

In deciding how many blocks of disk space to allocate to the system swap area, you should take into consideration: 1) the number of users you will be running, 2) the size of the programs and data bases that will be used, and 3) the total amount of disk space available for allocation.

The `/usr` file system will be allocated whatever disk space remains after space has been allocated for the other file systems and the swap area.

When you are *prompted*: "Enter Device,Unit,Size in 512- byte blocks: ", note that here "device" means the logical name of the file system, that "unit" is "0", and that your *response* should include a single space, or comma, between each of the requested pieces of information.

If you attempt to increase the size of any file system or the swap area, you will be told:

Allocation exceeds unit 0 size by ...

Note that this message is for information purposes only. In general when you change the size of any file system or the swap area, `divvy` automatically adjusts the sizes of the other allocations to insure that the total allocated never exceeds what is available.

If you choose to allocate blocks for the **/tmp** file system, after you have completed the installation procedure, you must change **/etc/fstab** to include the entry:

```
/dev/dsk/0s3 /tmp
```

so that the **/tmp** file system will be auto-mounted.

Alternately, you may choose to mount the third file system, (designated as **/tmp** by **divvy**), on another path such as **/usr1**. If you wish this file system to be auto-mounted at boot time, you must add the following entry to **/etc/fstab**:

```
/dev/dsk/0s3 /usr1
```

3.1 If you do not wish to use the default disk space allocations and **ONLY** wish to change the space allocation for the **/root** or **/usr** file system of the UNIX partition, you are asked:

1. *PROMPT* Do you wish to change any allocations?
(y or n):
RESPONSE y
2. *PROMPT* Enter Device,Unit,Size in 512- byte blocks:
RESPONSE Set the size of the **/root** file system to the size that you wish to be allocated for it (the **/usr** file system will be automatically allocate whatever space is not allocated to either **/root** or **/swap**)
3. *PROMPT* Do you wish to change any allocation?
(y or n):
RESPONSE n

Continue the installation procedure at Step 9 (Continuing the Use of the **divvy** Utility — Making the File Systems) of this chapter.

INSTALLATION INSTRUCTIONS

3.2 If you DO NOT wish to:

- A) use the default disk space allocations, and
- B) allocate space for a /tmp file system,

but you DO wish to:

- A) change the space allocation for the system swap area, and
- B) possibly changing the /root or /usr file systems of the UNIX partition:

1. *PROMPT* Do you wish to change any allocation?
(y or n):
RESPONSE y
2. *PROMPT* Enter Device,Unit,Size in 512- byte blocks:
RESPONSE Set size of /swap to the size that you wish
to be allocated to the system swap area
3. *PROMPT* Do you wish to change any allocation?
(y or n):
RESPONSE y
4. *PROMPT* Enter Device,Unit,Size in 512- byte blocks:
RESPONSE Set the size of the /root file system to the
size that you wish to be allocated for it
(the /usr file system will be automatically
allocated whatever space is not allocated to
either /root or /swap)
5. *PROMPT* Do you wish to change any allocation?
(y or n):
RESPONSE n

Continue the installation procedure at Step 9 (Continuing the use of the **divvy** Utility — Making the File Systems) of this chapter.

If you do not wish to use the default disk space allocations, and you wish to allocate disk space for a **/tmp** file system AS WELL AS possibly changing the system swap area or the **/root** or **/usr** file system of the UNIX partition, you are asked:

1. *PROMPT* Do you wish to change any allocation?
(y or n):
RESPONSE y
2. *PROMPT* Enter Device,Unit,Size in 512- byte blocks:
ACTION Initially, you will need to create a **/tmp** file system which overshoots in size the **/tmp** file system you ultimately wish to create, so set the size of the swap area to the total number of blocks available minus twice the number of blocks that you wish to finally allocate to the **/tmp** file system
3. *PROMPT* Do you wish to change any allocation?
(y or n):
RESPONSE y
4. *PROMPT* Enter Device,Unit,Size in 512- byte blocks:
RESPONSE Initially, set the size of the **/root** file system to zero blocks.
5. *PROMPT* Do you wish to change any allocation?
(y or n):
RESPONSE y
6. *PROMPT* Enter Device,Unit,Size in 512- byte blocks:
RESPONSE Request that the size of the **/tmp** file system be set to twice the size that you ultimately desire
7. *PROMPT* Do you wish to change any allocation?
(y or n):
RESPONSE y

INSTALLATION INSTRUCTIONS

8. *PROMPT* Enter Device,Unit,Size in 512- byte blocks:
RESPONSE Set size of **/swap** to the size that you wish to be finally allocated to the system swap area
9. *PROMPT* Do you wish to change any allocation?
(y orn):
RESPONSE y
10. *PROMPT* Enter Device,Unit,Size in 512- byte blocks:
RESPONSE Set the size of the **/root** file system to the size that you wish to be finally allocated to it
11. *PROMPT* Do you wish to change any allocation?
(y orn):
RESPONSE n

Continue the installation procedure at Step 9 (Continuing the use of the **divvy** Utility — Making the File Systems) of this chapter.

Notes

Any zero size allocation, such as the default allocation for the **/tmp** file system, does not show up on your screen.

Appendix 4

Adding new login names to the System

If you choose not to allow the SysVision user shell to add new login names, proceed with these instructions. Otherwise, see Chapter 3.

Instructions

In order to create a new user (login name) on your system:

1. *ACTION* login as root

Edit the `/etc/passwd` file to create an entry for the new login name, as follows:

```
PROMPT                      #
RESPONSE                  vi /etc/passwd
```

Using the entry for `guest` as a model, create a new entry with a null password field and a new user id field. Note that you can specify a shell pathname, such as `/bin/csh` for the C Shell, in the last field entry. This shell will be invoked at start-up. The default entry is `/bin/sh` (Bourne Shell). For further information, see `passwd (4)` in the Software Development System manual, Vol II.

To serve as an example for adding a new login name to the system, we will use the login name `bernie` in the following procedures:

2. *ACTION* Create a directory for the new login-name, `bernie`, as follows:

PROMPT #
RESPONSE `mkdir /usr/bernie`
3. *ACTION* Change owner of the created directory to the new login -name, `bernie`, as follows:

PROMPT #
RESPONSE `chown bernie /usr/bernie`
4. *ACTION* Copy over relevant files to the new user's directory:

PROMPT #
RESPONSE `cp /.profile /usr/bernie`

Note that the `.login` and `.cshrc` files are used by the C Shell.

```
ALTERNATE                  cp /.login /usr/bernie
RESPONSE                  cp /.cshrc /usr/bernie
```

Appendix 5

The drive type parameter which is referenced in the *setup* utility is an index into a table which is located in the ROM BIOS on your machine. If the table entry is not found by the *fdisk* utility, for types 1-14 the *fdisk* utility will assume the following hard disk parameters.

Table 1
Table of Standard Drive Types

type	heads	cyls	landing zone	write precomp
1	4	306	305	128
2	4	615	615	300
3	6	615	615	300
4	8	940	940	512
5	6	940	940	512
6	4	615	615	-1
7	8	462	511	256
8	5	733	733	-1
9	15	900	901	-1
10	3	820	820	-1
11	5	855	855	-1
12	7	855	855	-1
13	8	306	319	128
14	7	733	733	-1

The following table gives some manufacturer's drive parameters. If you do not have appropriate ROMs in your machine, you should attempt to match the number of heads to one of the standard types, and correct the values displayed by the *fdisk* utility to the correct values for your drive.

Table 2

Table of Manufacturer's Parameters

Manufacturer	Model Number	suggested type	capacity	cyls	heads	landing zone	write precomp
ATASI	3046	12	37.4	645	7	644	323
	3051	12	41	704	7	703	352
	3051+	12	42.6	733	7	732	368
	3085	4	68.0	1024	8	1023	-1
Computer Memories Inc. (CMI)	CM-6426-S	2	20.4	615	4	615	300
	CM-6426	2	21.3	640	4	640	256
	CM-6640	3	31.9	640	6	640	256
	CM-6853	7	42.5	640	8	640	256
Control Data Corp. (CDC)	CDC 94155-86	*	69.1	925	9	925	0
	CDC 94155-48	11	38.4	925	5	925	0
	CDC 94155-51	11	41.0	989	5	989	0
Maxtor	XT1065	12	53.3	918	7	918	-1
	XT1085	4	68.0	1024	8	1024	-1
	XT1105	*	83.8	918	11	918	-1
	XT1065	9	114.3	918	15	918	-1
Microscience	HH1050	11	44.0	1024	5	1024	-1
	HHx25	2	21.4	615	4	615	-1
Miniscribe	6032	10	25.5	1024	3	1024	512
	6053	11	42.5	1024	5	1024	512
	6032	12	59.5	1024	7	1024	512
	6032	4	68.0	1024	8	1024	512
Rodime	202E	2	21.3	640	4	640	0
	203E	3	31.9	640	6	640	0
	204E	7	42.5	640	8	640	0
Scagate	ST 4026	2	20.4	615	4	615	300
	ST 4038	8	30.4	733	5	733	300
	ST 4038M	8	30.4	733	5	733	-1
	ST 4051	11	40.5	977	5	977	-1
	ST 225	2	20.4	615	4	615	300
	ST 4096	*	76.5	1024	9	1023	-1

INSTALLATION INSTRUCTIONS

Manufacturer	Model Number	suggested capacity type	cyls	heads	landing zone	write prcomp	
Tandon (Microtek)	TM-703	8	28.8	695	5	695	-1
	TM-755	11	40.7	981	5	981	-1
	TM-703AT	8	30.4	733	5	733	-1
	TM-702AT	6	120.4	615	4	615	-1
Toshiba	MK56FB	**	68.9	830	10	830	-1
	MK54FB	12	48.2	830	7	830	-1
	MK53FB	8	34.4	830	5	830	-1
Tulin	TL-226	6	21.3	640	4	656	-1
	TL-240	3	31.9	640	6	656	-1
Vertex	V150	11	41	987	5	987	-1
	V170	12	57.4	987	7	987	-1
	V130	10	24.5	987	3	987	-1
	V185	12	59.5	1024	7	1023	-1
Priam	ID40	11	41.0	987	5	987	-1
	ID60	12	57.4	987	7	987	-1
	ID130	9	127.5	1024	15	1024	-1

*These drives do not have an entry with the same number of heads in the standard drive tables.

**This drive has a switch (#8) which allows heads 8 and 9 to be disabled. When this switch is set on this drive may be used as a type 7.

Chapter 3

SysVision

C-3

SysVision is a helpful group of menus and forms that aids the novice and System Administrator alike when using the UNIX system.

To begin using SysVision type `sysviz` `<return>` after your prompt. You can access SysVision from the C-shell and the Bourne shell.

After signing onto SysVision, the following menu is displayed:

```

mpmain(1a) SysVision by TaskForce(1b) 01/31/87(1c) 09:30(1d)

      User Functions                Administrator Functions
A   Applications Menu              *U  User & Group Menu (2)
C   Communications Menu            *F  Floppy Disk Menu
P   Printer Control Menu           *R  Save/Restore Menu
H   Hard Disk Functions Menu       *S  System Functions Menu
                                       *M  Machine Management Menu

Quit-^X  Help-^W  More Keys-^Y  (3)
Select Function by entering letter or using arrow keys and pressing
RETURN (4)

```

This is referred to as the main menu or system entry point. All the SysVision menus have the same format which consists of:

1. TITLE LINE

The title line is the first line on the menu and consists of four fields. The menu's name (1a) is displayed in the upper left corner, its title (1b) is centered and high-lighted, and the system date (1c) and system time (1d) are displayed in the upper right corner.

2. MENUS AND COMMANDS AVAILABLE

Each menu has a list of commands that are available. Select by pressing the letter in front of the option, or use arrow keys to position the arrow next to the option and press RETURN. Some functions are reserved for the exclusive use of the system administrator (root) and other users are denied access to them. The menu program will prevent you from moving the arrow next to these options or selecting them with their letter. They *usually* are displayed in a different color or intensity on your screen. They are denoted in this documentation by an asterick (*) next to the option. Note that all of the functions in the right column of the previous menu are restricted.

3. CONTROL KEY FUNCTIONS AVAILABLE TO USE ON THIS SCREEN

NOTE: ^ is the symbol for the control key, located on the left side of your keyboard. To enter a control key function, hold down the control key (like a shift key) and then press the letter next to it. For example, to enter ^X (read Control X), hold down the control key and strike the X key. The shift key does not have to be depressed as there is no difference between ^X and ^x.

On the main menu above the control key functions available are ^X, ^W and ^Y. Below is an additional list of control key functions that are used by the menu.

KEY	FUNCTION	DESCRIPTION
^X	Quit	Exit SysVision. Will ask Y/N.
^W	Help	Additional information about a menu option. Position the arrow at the option in question and press the help key (^W) and a help window will replace the menu. More than one page of help is signified by a plus sign (+) in the bottom right corner of the help window. You may advance the text by pressing the Roll Down key (^D). You may re-display the previous page by pressing the Roll Up key (^U) when the plus sign is displayed in the upper right corner of the help window. Press Exit (^E) to clear the help screen and re-display the menu.

^Y	More Keys	Additional control keys and their descriptions. Press Exit (^E) to clear the help screen and redisplay the menu.
^E	Exit	This will exit you out of any special mode you may be in.
^P	Previous Menu	This will display the previous menu. If you are using the main menu, it will ask you if you wish to quit.
^D	Roll Down	More information is available. Only available in help windows when the plus sign (+) is displayed in the bottom right corner.
^U	Roll Up	Review previously displayed information. Only available in help windows when the plus sign (+) is displayed in the upper right corner.
/	Change Directory	This function displays your current working directory and allows you to select a new directory.
	Shell Command	This function allows you to execute a single Bourne shell command. An input area is opened on the message line and you may type the command you wish to execute. When RETURN is pressed, the screen is cleared and the command is executed.
^F	Execute	If it is shown on the screen as an available control key, it must be entered to start the function.

4. MESSAGE LINE

Additional information, prompts, and warnings are displayed here. It is also replaced by the input area for the Change Directory and Shell Command functions.

Select the letter next to the function you wish to perform and proceed to the corresponding section in the manual.

COMMUNICATIONS MENU

INTRODUCTION

The Communications Menu allows you to send and receive messages, or type messages directly onto another terminal. It also allows you to "log on" to other computers, allowing your PC to function as a remote workstation.

```

mpcomm   COMMUNICATIONS   01/31/87 09:30

      Local                      Remote
S  Send Mail                    C  Log onto another system
R  Receive Mail                 M  Mail to another system
W  Write a message to a user    U  List of known systems
H  Who is logged in?

Quit-^X  Help-^W  More Keys-^Y

Select Function by entering letter or using arrow keys and pressing
RETURN
    
```

Select the letter or character for the function you wish to perform and proceed to the corresponding command in this section.

S - SEND MAIL

This command is used to send a message, such as an interoffice memo, to another user on the system. You can also have copies of your message sent to other users.

Send to : _____

Subject: _____

Date: _____

Message: _____

Send copies to: _____

To use Send Mail, simply fill in the blanks and use either the RETURN key or the down arrow key to go from line to line.

Enter the user's login name on the Send to: line. If you do not know the user's login name, press ^W (Help) for a list of users on the system.

Next, enter the subject of your message, as you would on a memo. This is a short phrase that describes the text of the memo.

Enter the date in month, day, year format, for example: 12/31/86.

Enter your message. The field will automatically extend up to seven more lines for your message.

If you want to send copies of your message to other users who are on the system, enter their LOGIN NAMES on the line. Separate the names with a space or a comma. Again, if you are unsure of the user's names, press ^W (Help) for a list of users on the system.

Press ^F to execute the command and send the memo.

R - RECEIVE MAIL/READ MAIL

This command is used to receive messages/interoffice memos sent to you from other users.

Display mail without asking questions? _____

Display mail in First In - First Out order? _____

Name of alternate mail file _____

Normally, when displaying your mail, a question mark (?) will appear on the screen after each item of mail you receive. This prompt allows you to choose what you would like to do with that item. To use any of the options, answer N to the first question. If you would simply like to view the messages displayed on the screen, one right after the other, answer Y and you will not be prompted after each message.

Key	Action
<RETURN>	Go to next item
+	Go to next item
d	Delete and go to next item
p	Print item (on screen) again
-	Go back to previous item
s[FILE]	Save the item in the file given [FILE] (mbox is the default)
w[FILE]	Save the item in the file given [FILE] without the header information
m[USER]	Mail the message to [USER](s) named
q	Put undeleted mail back in the mail file and quit
x	Put all mail back in the mail file and quit

Mail is usually displayed *last-in, first-out* order. This means that your newest messages will be displayed first, and your oldest last. If you would like to have your mail displayed in *first-in, first-out* order (oldest first, newest last), answer **Y**.

If you have, in previous receipts of mail, used the options s[FILE] or w[FILE] (described above), you will have saved items of mail in alternate mail files (mbox by default). You can again retrieve messages from these alternate files by specifying their name in the third blank on the screen.

Press ^F to execute the command. Your mail will be displayed on the screen. If you did not respond with a **Y** to the first question, a question mark will be displayed after each message. Please select the appropriate response from the above table.

W - WRITE A MESSAGE TO A USER

This command allows you to send a message directly to the screen of another user. You might use this to send information or a warning quickly.

Enter the name of the person to write to _____

Indicate which terminal to write to if
the user is logged in more than once _____

On the first line, enter the person's **LOGIN NAME**. If you want to know who is logged in, press ^W (Help) to list all users and the terminals they are using. If that user has logged in to more than one terminal, fill in the name of the terminal they are currently using in the second input field.

Press ^F to execute the command.

The screen will be cleared, and the text will appear:

Type the message to be sent (terminate by pressing ^D)

Begin typing in the message. When you are through, press RETURN to take the cursor to the beginning of a blank line, then press ^D.

When you are through you will be prompted to press RETURN to go back to the menu.

NOTE: There is a way a user can prevent you from writing onto their terminal. If this is the case, the message **Permission Denied** will be displayed on your screen. For more information, please refer to the **mesg(1)** command in the System V/AT Runtime System Manual.

H - WHO IS LOGGED IN?

This command allows you to find out quickly who is logged in to the computer system, which terminal they are using, and what time they logged in. It is an informational screen only. An example screen showing the format of the display follows (your output will be different):

Login Name	Terminal	Date Logged In
root	console	Feb 8 16:24
paul	cons2	Feb 8 14:44
bill	tty0	Feb 8 13:21
markc	tty1	Feb 8 18:55

You will be asked to press RETURN to redisplay the menu.

C - LOG ONTO ANOTHER SYSTEM

This command allows you to call and log onto another computer system. You can use your terminal as a workstation for another computer, not just the one you normally use.

System to be called _____

Enter the name of the system you wish to access. If you don't know the name of the system press ^W (Help) for a list of computer systems that are available for

SYSVISION

you to log onto. A list of system names and associated descriptions will be displayed in the help window.

Once you have entered the appropriate system name, press **^F** to execute the command.

After you have finished using the remote system, be sure to logout and then terminate the session by typing **~.<RETURN>** which will instruct your system to break off communications with the remote system. For more information, please refer to the **cu(1)** command in the System V/AT Runtime System Manual.

M - MAIL TO ANOTHER SYSTEM

This command is used to send a message to a user on another system. You can also have copies of your message sent to other users.

Send to : _____
(address as "sysname!sysname!user")

Subject: _____

Date: _____

Message _____

Send copies to: _____

Enter the user's name on the "Send to:" line. Notice that you can address the user as "sysname!sysname!user". Each system name, separated by a "!" (known as a "bang"), indicates a step in the network of interconnected systems which must be traversed to reach the user.

For example: `tsc!baysys!markc`

This address indicates that first the system named "tsc" must be called which in turn will call the system named "baysys" on which the user "markc" can be found.

You can press **^W (HELP)** for a list of systems immediately accessible to your system.

Next, enter the subject of your message.

Enter the date in month, day, year format, for example: 12/31/86.

Enter your message.

If you want to send copies of your message to other users who are on other systems, enter their "names" the same way as above. Separate the names with a space or a comma. You may send copies to users on your own system as well as on other systems. For users on your local system, you must omit the system name.

Press ^F to execute the command and send the mail.

U - DISPLAY A LIST OF KNOWN SYSTEMS

This command allows you to find out quickly what computer systems are available to log onto. The screen will display the available systems by System Name and Description. This is an informational screen only. An example screen showing the format of the display follows (your output will be different):

System Name	Description
tsc	TaskForce Software Corporation
uport	Microport Systems Inc.

You will be asked to press RETURN to redisplay the menu.

HARD DISK MENU

INTRODUCTION

The Hard Disk Functions Menu allows you to perform many "housekeeping" tasks for the hard disk. It enables you to make a new directory, change to another directory, or remove a directory. It also allows you to move files around on the hard disk, copy them, rename them, or remove them entirely.

```

mpdisk      HardDisk Functions    01/31/87 09:30

      Directory Functions          File Functions
D  Display Current Directory      P  Display a File
L  List Current Directory         M  Move a File
/  Change Directory              R  Rename a File
M  Make a Directory              C  Make a Copy of a File
R  Remove a Directory            Z  Copy a File to a Directory
                                   X  Remove a File

Quit-^X Help-^W More Keys-^Y Previous-^P

```

Select the letter or character for the function you wish to perform and proceed to the corresponding command in this section.

D - DISPLAY CURRENT DIRECTORY

This command allows you to display the name of the directory in which you are currently positioned. It is useful in reminding you of your location in the system's directory hierarchy.

When you press D from the menu, the screen will be cleared and the name of the directory at which you are currently positioned will be displayed at the bottom left. Simply press RETURN to redisplay the menu.

L - LIST CURRENT DIRECTORY

This command allows you to display the contents of the current directory. You can specify how detailed you want this information to be.

Detailed listing of files? _

Display file types? _

Answer Y or N to these two questions. The default is N.

A detailed listing of files includes the file's ownership, its size in bytes, and the date on which it was created. If you do not select a detailed listing of the file, only its name will be displayed.

File types are displayed, if they are requested, as special characters at the end of the file name. The types are:

"/" = directory "*" = executable file " " (blank) = text file

Press ^F to execute the command

/ - CHANGE DIRECTORY

You can change your position in the hierarchy of directories from any menu. The directory in which you are presently positioned is referred to as your "current directory". By simply pressing the "/" (slash) key, you can change your position from one directory to another.

When you press the "/", two lines appear at the bottom of the screen. The first is the name of your current directory (your current position).

Next appears a prompt line which reads:

Enter new directory name _____

Type in the name of the directory you wish to move to and press RETURN. You will then be moved to this new directory. If, however, the directory you specified does not exist, or was entered incorrectly, you will receive the message:

"Directory does not exist"

"Could not perform requested function. Press RETURN to continue or ^E to exit."

You can simply press RETURN to try again, or press ^E to abandon your attempt to change your current directory position.

Examples:

You are located in /usr/tmp. You move to directory /usr/acct by pressing the "/" key, and then typing "/usr/acct" and pressing RETURN.

Suppose further that after moving to /usr/acct, you wish to change to a subdirectory that is called /usr/acct/bob. All you need to enter is "bob".

To change back to the former directory, you must enter "/usr/acct".

M - MAKE A DIRECTORY

This command allows you to create new directories (or subdirectories).

Directories to be created _____

You will generally only be creating new directories as subdirectories of your home position. To see a list of the files and directories contained within your current directory Press ^W (Help).

Enter the name of the directory to be created. For example, enter "proposals". You will create a subdirectory contained within your current directory. In it you might create or store proposals.

Enter ^F to execute the command.

R - REMOVE A DIRECTORY

This command allows you to remove unused or unwanted directories. In order for a directory to be removed, it must be empty. It must not contain any files or subdirectories.

Directories to be removed _____

Force the removal of all files and N
subdirectories within the directory
to be removed

You will often be removing subdirectories of your current directory. It is therefore often helpful to press ^W (Help) to display the contents of your current directory.

Enter the name of the directory to be removed.

At times you will wish to remove a directory along with its contents. You can cause the system to remove a directory and all the files and subdirectories it contains by answering **Y** to the next prompt. Notice that this blank is filled in with a default value of **N**. This will help to remind you that you must **USE CAUTION** when removing files in this manner.

Enter **^F** to execute the command.

P - DISPLAY A FILE

This command allows you to display a text file on the screen. The file will be displayed one page or screen full at a time.

Files to be displayed _____

If you want to select a file from within your current directory press **^W (Help)** to display a list of these file names.

Enter the name of the file to be displayed. You can enter more than one file name by simply separating their names with spaces or commas. The files will be displayed sequentially.

Enter **^F** to execute the command.

Each file is displayed as a series of pages (screen fulls) of text. At each you will be prompted to press **RETURN** to proceed to the next page. If you do not wish to continue paging through the file, press **Q** and then press the **RETURN** key. The display of the file will be interrupted and you will be prompted to press **RETURN** to go back to the menu.

V - MOVE A FILE

This command allows you to move a file from one directory to another. This is useful when you want to place files of the same type together in one directory.

Name of file(s) to be moved _____

Target directory name _____

Enter the name or names of the files that you want to move. If you have more than one file to move, separate the file names with a space or a comma. Enter the name of the directory that is the destination (or target directory) of the files.

Enter ^F to execute the command.

N - RENAME A FILE

This command allows you to change the name of a file.

Current file name _____

New file name _____

If you are renaming files within your current directory, you can press ^W (Help) to display a list of their current names.

First enter the file name that you want to change.

Next enter a unique new name for the file.

Enter ^F to execute the command.

C - MAKE A COPY OF A FILE

This command allows you to make a duplicate of a file.

From file: _____

To file: _____

If you are copying a file in your current directory, press ^W (Help) to display a list of those file names.

First enter the name of the file that you wish to copy.

Next enter the file name that you wish to give the new copy. It should be a new name, not one that already exists. If you do enter an existing file name, you will be warned with the message,

Target file already exists — Cannot destroy

Simply enter a unique name for the new file and press RETURN.

Enter ^F to execute the command.

Z - COPY A FILE TO A DIRECTORY

This command allows you to make a duplicate of a file and store it in another directory. Both files will have the same name, but be contained within different directories.

Name of file(s) to be copied _____

Target directory name _____

If the files you are copying from are contained in your current directory, press ^W (Help) to display a list of their names.

Enter the name(s) of the files that you want to copy. If you have more than one file to copy, separate the file names with a space or a comma. Enter the name of the directory that is the destination or target of the files.

Enter ^F to execute the command.

X - REMOVE A FILE

This command allows you to remove files from the disk. Since deleting files accidentally can be a problem, safeguards have been placed in this command.

Ask for confirmation before deleting
each file? _____

Remove files for which you have no
write permission without asking
for confirmation? _____

Files to be removed _____

By answering yes "Y" to this first prompt, you will cause the system to ask for confirmation from you before removing each file. Each file name will appear on the screen and you cause its removal by entering a "y". This makes it possible for you to change your mind about removing a file.

SYSVISION

Sometimes in UNIX, files are created to which your access is limited in different ways. For instance, you may be allowed to look at file, but you cannot change it. (These files are often special tables that a program uses or some other information that is important to keep untouched.) Normally when removing these files, you will be asked for confirmation first. By answering "Y" to the second prompt on this screen, you will cause the system to remove these kinds of files without asking for confirmation. **USE CAUTION WHEN SELECTING THIS OPTION.**

Enter the name(s) of the file(s) to be removed. Separate each file name with a space or a comma.

If you are removing files that are in your current directory press ^W (Help) to display a list of these file names.

Enter ^F to execute the command.

PRINTER CONTROL MENU

INTRODUCTION

The Printer Control Menu allows you to control the UNIX print spooler. You may send a file that you want to print to a printer queue. When the printer becomes available, the file is printed. This allows you to make print requests any time and have the system manage the availability of the device.

Printer Control allows several printers to be shared among many users. Printers can be grouped together in "classes", for example, letter quality printers in one class, dot-matrix in another. Print requests can be queued by class, allowing your file to be printed on the next available printer in the class you requested.

Print requests can be cancelled if not needed. Printers may be stopped if there is a jam or a paper out problem. Printers can be prevented from accepting requests if they are out of service and can accept them again when the problem has been resolved.

```

mpadmpl      PrinterControl  01/31/87 09:30

      UserFunctions                Spooler Control

P  Print a File                    C  Cancel Spool Entry

S  Display Printer Status          E  Enable a Printer

      Scheduler                    F  Disable a Printer

L  Start Printer Scheduler          J  Accept a Printer Request

T  Stop Printer Scheduler           R  Reject a Printer Request

Quit-^X Help-^W More Keys-^Y Previous-^P
  
```

Select the letter or character for the function you wish to perform and proceed to the corresponding command in this section.

P - PRINT A FILE

Press F to Print a File. This command allows you to print a text file on the printer.

Destination (printer or class) name _____
Send message when printing complete? _
 m - via mail system
 w - notify immediately
 (write msg. on your screen)
Number of copies to print 1
Title for banner page _____
Name of the file to be printer _____

First enter the name of the printer (or class of printers) at which you want the file to be printed. If you don't know an appropriate name, press ^W (HELP) to display a list of printer names.

Often you wish to be notified when your file has finished printing. If so, enter "M" to be notified via the mail system, or "W" to be notified more immediately by a message written directly to your screen.

Enter the number of copies you wish printed. The default is one copy.

Enter the text which will appear on the banner (leading) page of your output. (The interface program associated with some printers does not accomodate a banner page, so this option will not work with all printers.)

Enter the name(s) of the text file(s) you wish to print. You can press ^W (HELP) for a list of the files in your current directory. If you specify more than one file, separate their names with a space or a comma.

Enter ^F to execute the command.

S - DISPLAY PRINTER STATUS

Press S to Display Printer Status. Information about the printers you have available will appear on the screen.

Example:

```

scheduler is running.
system default destination: p1
device for lp: /dev/lp
device for p1: /dev/tty0

lp accepting requests since Jan 25 11:13
p1 not accepting requests since Jan 26 17:50
  A/P Checks mounted

printer lp disabled since Jan 26 18:01
  januned
printer p1 is idle, enabled since Jan 25 02:38

```

From this display, you can determine:

1. The status of the print scheduler. The scheduler is the program that controls the printers. If it is active, enabled devices will print spool entries that are in the print queue. If the scheduler is off, no printing will occur at any device.
2. The system default printer. This is the destination that user output is routed to if a specific printer is not requested.
3. Which print devices are connected to your system and their status.
4. Printer's acceptance status. Printers that accept requests will allow the user to route output to them using the `lp` command. Printers that reject requests will notify the user that they are not available when they try to route output to their destination. Control over acceptance status is set by the `accept` and `reject` commands which are explained in this section. For further information on these commands, you may also reference the System V/AT Runtime System manual.
5. Printer's enable status. Printers that are enabled are actively controlled by the print scheduler program. They will print output as it becomes available in the print queue. Control over enable status is set by the `enable` and `disable` commands which are explained in this section. For further information on these commands, you may also reference the System V/AT Runtime System manual.

Press RETURN to redisplay the Printer Control Menu.

L - START PRINTER SCHEDULER

The Scheduler controls all of the printers that are attached to your system. As each job is completed the scheduler takes the next job request in the queue and routes it to the appropriate printer to be printed. While the Scheduler is running, print jobs that are routed to an enabled device will be printed. If it is not running, jobs will not be printed on any printer.

The Printer Scheduler is started automatically each time the system is turned on. However, if the Scheduler has been stopped it must be restarted before any printing will occur.

To Start the Printer Scheduler press L.

T - STOP PRINTER SCHEDULER

Occasionally it is necessary to reconfigure the printer system using commands that cannot be executed unless the Scheduler is inactive. The Stop Printer Scheduler command prevents the Scheduler from starting any new print jobs. Any requests currently printing will be completed.

Stopping the scheduler prevents the computer from attempting to use all printers. Specific printers may be stopped using the disable command.

To Stop the Printer Scheduler press T.

C - CANCEL SPOOL ENTRY

Press C to cancel requests in the print queue.

This command is used to remove one or more print jobs from the queue. It may be used before a job begins to print or after it has already started.

Request ID(s) to be cancelled _____

Cancel request now printing on printer _____

The Cancel command can be used one of two ways.

1. To cancel specific spool requests:

Enter the request id for the entry you wish to cancel. If necessary, press the ^W (Help) to display a list of your spool entries. The help window will contain a display in the following format: (your information will be different)

Request ID	User Name	Size	Date Submitted
p1-1117	paul	1167	Feb 8 09:17 on p1
p1-1119	paul	2107	Feb 8 09:23

Leave the second field blank and press ^F to execute the command.

2. To cancel the job that is currently printing on a particular printer.

Enter the printer name, e.g. p1, that is currently printing the job that you want to cancel. If necessary, press the ^W (Help) to display the list of printers that are available, and their status. The help window will contain a display in the following format: (your information will be different)

```
printer lp disabled since Jan 26 18:01
  januned
printer p1 now printing p1-1117, enabled since Jan 25 02:38
```

Leave the first field blank and press ^F to execute the command.

E - ENABLE A PRINTER

The enable command activates the printer to start printing print jobs that are in the queue.

Printer(s) to be enabled _____

Enter the names of the printers you want to activate. To enter more than one, put a space or a comma between each printer name. If necessary, press ^W (Help) to display a list of printers attached to your system and their current status.

Press ^F to execute the command.

F - DISABLE A PRINTER

The disable command deactivates a printer from printing spool requests. This will prevent the printer from printing any jobs that are in the queue. This command is especially useful if a paper jam or hardware problem occurs.

If a printer is in use at the time it is disabled, then the request that was printing will be reprinted in its entirety when the printer is enabled. Print requests can still be routed to a printer that is disabled. They go into the queue and will print when the printer is enabled.

Reason for disabling printer _____

Cancel any request currently printing ___

Printer(s) to be disabled _____

Enter a reason for disabling the printer, such as paper jam, hardware problem, paper out, etc. This will be shown on the screen status report, and will let other users know what has happened to the printer(s).

Answer Y or N after Cancel any request currently printing. If you answer Y, the entry currently printing will be removed from the print queue. If you answer N, the entry will be reprinted in its entirety when the printer is enabled.

Enter one or more printer device names, with a space or comma separating them. If necessary, press ^W (Help) for a list of printers available.

Press ^F to execute the command.

J - ACCEPT PRINTER REQUESTS

Accept Printer Requests allows the named printer destinations to begin accepting user print requests. Destination is the name of a printer or a class of printers.

The accept command allows users to reference a printer and route output to it. The enable command determines whether the print scheduler will initiate printing at the device.

Destination(s) to begin accepting requests (printers or class names) _____

Enter one or more printer or class names with a space or comma between them. If necessary, press ^W (Help) for a list of available printers and class names.

Enter ^F to execute the command.

R - REJECT PRINTER REQUESTS

Reject Printer Requests prevents the scheduler from accepting requests for the named print destinations. For example, if too large a backlog has built up at a certain printer or if the printer has been removed, it would be necessary to prevent user requests from being routed to that destination.

Any requests that are in the queue will remain there until they are printed, cancelled, or moved to another queue. When the condition that caused the reject has been corrected, use the accept command to allow requests to be received at that destination again.

Reason for causing destination to reject
lp requests _____

Destination(s) to reject
(Printer or class names) _____

Enter a reason for rejection of lp requests, such as too many requests or printer out for repair. This will be shown on the screen status report, and will let other users know what has happened to the printer(s).

Enter one or more printers/class names with a space or comma between them. If necessary, press ^W (Help) for a list of printers/class names available.

Enter ^F to execute the command.

USER AND GROUP MENU

INTRODUCTION

The User and Group Menu allows the System Administrator to add new user accounts and groups to the system, or change or delete existing user accounts and groups. The administrator can also change passwords for the accounts or force all users to change their password the next time they log in.

```

mpadmusr      AdministerUsers      01/31/87 09:30

      User Maintenance                Group Maintenance
U  Add a User Account                G  Add a Group
C  Change a User Account              I  Delete a Group
D  Delete a User Account              Setup
P  Change a User Password             M  Modify addusr defaults
F  Force all new Passwords

Quit-^X Help-^W More Keys-^Y Previous-^P

Select function by entering letter or using arrowkeys and pressing
RETURN
  
```

Select the letter or character for the function you wish to perform and proceed to the corresponding command in this section.

U - ADD A USER ACCOUNT

This command allows the System Administrator to add new user accounts to the system. The Add User command can be customized for your installation by running the Modify-Add User command which is explained at the end of this section. You may determine what values will be prefilled for the Group ID and Shell fields as well as which directory is used for creating home directories for new users. The values that may appear on your screen may be different from those that follow. For more information, please refer to the Modify Add User command at the end of this section.

User ID number	_____
Group ID	1_____
User name and phone number	_____
Home directory	_____
Shell	bin/sh
Account Name	_____

Enter a user id number, or the system will automatically assign the next available number. Leaving this field blank and letting the system assign the next available number is recommended. This will prevent you from inadvertently assigning duplicate User ID numbers.

Enter the group id number. The default group id is displayed and you may change it if you wish. If you want to know what the available group ids are, press ^W (Help) and a list showing GROUP NAME, ID, and MEMBERS will be displayed in the help window. Select the id corresponding to the group of which you wish the user to be a member.

Enter the user name and phone number. This is considered a comment field and is used for informational purposes only. All characters except a colon (:) are accepted.

The user's home directory is where he will be placed when logging in. If no directory name is entered here, the system will create a subdirectory in the default home directory, which is displayed at the bottom of the screen when the cursor is in this field. It is initially setup as /usr/acct and may be modified by using the Modify Add User command.

There are two main shells that are used in UNIX; the Bourne Shell and the C Shell. Enter either /bin/sh, for Bourne Shell or /bin/csh, for C Shell or the full path name of the user's initial program.

Each user must have a password in order to log on to the system. The administrator can assign a password to the account by entering Y to "Assign an initial password?", or the user will assign his own password when he first logs onto the system. The administrator has more control of the account if he first assigns the password.

Enter the Account Name, which is also referred to as the LOGIN NAME. If an account already exists for that name, an error message will appear at the bottom of the screen: "Account already exists for that name". Press ^C to clear the line and enter a new, unique name.

Enter ^F to execute the command. It will take a few seconds and a message "Working..." will display on the screen.

If you answered Y to assigning a password to the account the screen will be cleared and you will be prompted:

Assigning password for user: ACCOUNTNAME

New Password:

(ACCOUNTNAME is the name that you previously assigned the user.) Enter the password, then re-enter it when prompted. You will notice that the password is not displayed when you type it. To ensure that you typed it correctly, the computer requires that you retype it and compares it against your first response. If there are any differences, it will required you to enter it again. When complete, the USER and GROUP MENU will return.

C - CHANGE A USER ACCOUNT

This command allows the System Administrator to change information on a user account.

Account Name to be changed _____

Enter the Account Name to be changed. For more information, press ^W (Help) to display a list of users. The information is displayed in the format: LOGIN NAME, ID, GROUP, AND COMMENT. The LOGIN NAME is the same as the ACCOUNT NAME. COMMENT is the information that was entered as the name and phone number.

Enter ^F to execute the command. The system will retrieve the information for that user and in a few seconds a new screen will be displayed.

Group ID	_____
User name and phone number	_____
Home directory	_____
Shell	_____
Account Name	_____

The lines will contain the information that was entered when the user account was first added. Move to the line or lines that are to be changed. Enter the new information.

Enter ^F to execute the command. When complete, the USER and GROUP MENU will return.

D - DELETE A USER ACCOUNT

This command allows the System Administrator to delete a user account and optionally his home directory and all files and subdirectories contained within it.

Remove home directory and all files? _

User name(s) _____

If all files belonging to the user account(s) should be totally eliminated answer **Y** to the first prompt. This will recursively delete the user's home directory and all files and subdirectories contained within it. **IF YOU ELECT TO SELECT THIS OPTION, BACKING UP THE FILES BEFORE PROCEEDING IS HIGHLY RECOMMENDED.** If other users share the directory and files or you wish to preserve their information, answer **N**.

Enter the user name(s) that are to be deleted. If there is more than one name, separate the names with a space or a comma.

Enter ^F to execute the command.

If you selected **Y** for the first prompt, the screen will be cleared, and the following prompt displayed for each user name specified:

About to delete home directory: HOME DIRECTORY for user:
ACCOUNT NAME

Enter 'y' to continue or 'n' to bypass directory and file removal

Enter **y** to delete this user's files and directories and **n** to bypass delete and proceed with the next user. When complete, the **USER** and **GROUP MENU** will return.

P - CHANGE A USER PASSWORD

This menu allows the System Administrator to change a user's login password.

Login name _____

Enter the login name of the user you wish to change or press **^W** (Help) to display a list of users. The information is in the format: **LOGIN NAME, ID, GROUP, AND COMMENT**. The **LOGIN NAME** is the same as the **ACCOUNT NAME**. **COMMENT** is the information that was entered as the name and phone number.

Enter **^F** to execute the command. Immediately the screen will clear and you will be prompted:

New password:

Enter the new password, then re-enter it when prompted. If they do not match you will have to enter it again. When complete, the **USER** and **GROUP MENU** will return.

F - FORCE ALL NEW PASSWORDS

This command allows the System Administrator to force **ALL** the user accounts to select a new password the next time they sign on. This enables a feature of **UNIX** called password aging. The system password file is updated with a code that requires all users (except root and other secured administrative logins) to pick a new password the next time that they sign onto the system.

Are you sure you want each user to
select a new password? _____

The default is **Y**. Use caution when executing this command and make sure users are notified of your actions.

Enter **^F** to execute the command. When complete, the **USER** and **GROUP MENU** will return.

G - ADD A GROUP

This command allows the System Administrator to add groups to the system group file.

Numerical Group ID _____
 Group Members _____
 Group Name _____

C-3

Enter an id number for the group, or the system will search for the highest id that currently exists, increase it by one, and assign it to the group.

Enter the LOGIN NAMES of the users that are to be in the group. If necessary, press ^W (Help) to display a list of users. The information is in the format: LOGIN NAME, ID, GROUP, AND COMMENT. COMMENT is the information that was entered as the name and phone number. If more than one name is entered, separate the names with commas or blanks.

The Group Name is 1 - 8 characters long and must be unique (not already used by the system).

Enter ^F to execute the command. When complete, the USER and GROUP MENU will return.

I - DELETE A GROUP

This command allows the System Administrator to delete groups from the system.

Group Name(s) _____

Enter the name of the group(s) to be deleted. If necessary, press ^W (Help) to display a list of available groups, with the GROUP NAME, ID, and MEMBERS. If more than one group is to be deleted, separate the names with commas or blanks.

Enter the ^F to execute the command. When complete, the USER and GROUP MENU will return.

M - MODIFY ADDUSER DEFAULTS

This command allows the System Administrator to create defaults that are used when user accounts are added. This saves time when many of the users are being placed in the same main/parent directory, group, or shell.

Parent Directory for new accounts _____
Default Group ID _____
Default Shell _____

Enter the directory that is to be used as the parent directory, such as "/usr/acct." This will appear as the default on the message line (at the bottom of the screen) in the add user command. (If this directory does not exist, it will be created)

Enter the id of the default group. If necessary, press ^W (Help) to display a list of available groups, displaying the GROUP NAME, ID, and MEMBERS. New user accounts that are added will belong to that group unless you override this value when executing the add user command.

Enter the name of the shell you wish to be default. This may be either /bin/sh for the Bourne shell, /bin/csh for the C shell or the name of any executable program.

Enter ^F to execute the command. When complete, the USER and GROUP MENU will return.

NOTE: Once the defaults are changed, you need to exit SysVision and restart the program for them to take effect. This is because the values are stored in environment variables that are set at program startup time.

FLOPPY DISK MENU

INTRODUCTION

This menu allows you to prepare and use floppy disks as extensions your computer's hard disk. It allows you to construct file systems on a floppy disk which UNIX treats the same as a file system on the hard disk. First the floppy disk must be formatted, then a file system must be made on it. In order to access a file stored on a floppy disk file system, it must be mounted, then when finished, unmounted. This menu will also allow you to copy files to and from the floppy disk file system while it is mounted.

```

mpflop   Floppy DiskFunctions  01/31/87  09:30

Floppy Media Preparation          Copy to/from Mounted Floppy

F  Format a Floppy                X  File/Dir to Floppy
S  Make a Floppy File System      Y  File/Dir to Hard Disk
C  Check a Floppy File System     Other Floppy Media Commands
M  Mount a Floppy File System     D  Duplicate a Floppy
U  Unmount a Floppy File System

Quit-^X Help-^W More Keys-^Y Previous-^P

Select function by entering letter or using arrow keys and pressing
RETURN
  
```

Select the letter or character for the function you wish to perform and proceed to the corresponding command in this section.

F - FORMAT A FLOPPY DISK

This command allows you to format a floppy disk, which prepares it for use on your floppy disk drive.

Name of device containing floppy
to be formatted

First you must select the name of the device which contains the disk. You may press ^W (Help) to display the list of devices available. Enter the two character name of the device you wish to use. A table of devices is listed below.

<u>Name</u>	<u>Device</u>	<u>Description</u>
0h	/dev/rdisk/fd096	1.2 MB High Density Floppy in Drive 0
0d	/dev/rdisk/fd048	360 KB Double Density Floppy in Drive 0
1h	/dev/rdisk/fd196	1.2 MB High Density Floppy in Drive 1
1d	/dev/rdisk/fd148	360 KB Double Density Floppy in Drive 1

Make sure the floppy disk is loaded correctly and the door closed.

Enter ^F to execute the command.

The screen will display the message that the formatting process is taking place, and on which device. When formatting is complete, you will be prompted to press RETURN to go back to the menu.

S - MAKE A FLOPPY FILE SYSTEM

This command allows you to create a file system on a floppy disk allowing the floppy to be used as an extension of the hard disk system.

Name of device for making file system __

You may press ^W (Help) for a list of the devices that are available. Enter the two character name corresponding to the the device in which the floppy disk has been placed.

Enter ^F to execute the command.

The screen will display a message that the file system is being created. You will be prompted to press RETURN when it is complete.

C - CHECK A FLOPPY FILE SYSTEM

This command allows you to check a floppy file system that you have created earlier. It lets you make sure that the disk has no errors after a period of prolonged use.

Name of device for checking file system __

Next enter the name of the device in which the floppy disk has been placed. Press ^W (Help) for a list of the devices that are available. Enter the two character name corresponding to the the device in which the floppy disk has been placed.

Enter ^F to execute the command.

The screen will display a message that the file system is being checked. You will be prompted to press RETURN when it is complete.

For more about checking file systems, refer to `fsck(1)` in the System V/AT Runtime System manual.

M - MOUNT A FLOPPY FILE SYSTEM

This command allows you to mount a floppy file system so that you can access the programs or files that are on that disk. "Mounting" informs the system that a device (a disk or a tape) is now available for use.

Name of device with floppy to be mounted

Directory on which to mount the floppy

Mount file system as read-only? N

First you must select the name of the device in which the floppy disk has been placed. You may press ^W (Help) for a list of mountable devices. Enter the two character name corresponding to the the device in which the floppy disk has been placed.

Next, enter the directory on which to mount the floppy. Normally, the directory `/mnt` is used. This is a directory created specifically for the purpose of mounting.

The file system can be mounted as read-only preventing the user from making changes to the files that are on the floppy disk. Simply answer Y(es) to the next prompt to mount the floppy file system in a "read-only" manner.

Enter ^F to execute the command.

A message will display on the screen that the file system is being mounted. When it has completed, you will be able to access the files on the floppy disk.

U - UNMOUNT A FLOPPY FILE SYSTEM

This command allows you to unmount a floppy file system when you are through using the programs or files on that disk.

Name of device to be unmounted? ___

You may press ^W (Help) for a list of mountable devices. Enter the two character name corresponding to the the device in which the floppy disk has been placed.

Enter ^F to execute the command.

A message will display on the screen that the device has been unmounted.

X- [COPY] FILE/DIR TO FLOPPY

This command allows you to copy files and/or directories from the hard disk to the currently mounted floppy file system disk.

Name of device at which floppy is
currently mounted ___

Files/directories to be copied _____

First you must enter the name of the device in which the floppy disk has been placed. You may press ^W (Help) for a list of mountable devices. Enter the two character name corresponding to the the device in which the floppy disk has been placed.

On the next line, you may press ^W (Help) again, to display the contents and name of the current directory. Enter the file names and/or directories to be copied to the floppy disk. If there is more than one file, separate the files names with a space or a comma.

Enter ^F to execute the command.

The screen will display the file names as they are copied.

Y - [COPY] FILE/DIRECTORY TO HARD DISK

This command allows you to copy files and/or directories from the currently mounted file system to the hard disk.

Name of device at which floppy is currently mounted	_____
Files/directories to be copied. (located on the mounted floppy)	_____
Target directory on hard disk to which files will be copied	_____

First you must specify the name of the device on which the floppy file system has been mounted. You may press ^W (Help) for a list of mountable devices. Enter the two character name corresponding to the device in which the floppy disk has been placed.

On the next line, you may press ^W (Help) again, to display the contents and name of the current directory. Enter the file names and/or directories to be copied from the floppy disk file system. If there is more than one file, separate the files names with a space or a comma.

Enter the directory name on the hard disk to which you want to copy the files. That is the target directory. If no directory name is entered the files will be copied into your current directory.

Enter ^F to execute the command.

The names of the files will be displayed on the screen as they are being copied.

D - DUPLICATE A FLOPPY

This command will allow you to make a duplicate of a floppy disk. It is not necessary to first mount a floppy in order to duplicate it. It will contain exactly the same information as the original disk.

Name of device for floppy duplication _____

You may press ^W (Help) or a list of devices. Enter the two character name corresponding to the device in which the floppy disk has been placed.

SYSVISION

Insert the "original" disk into the disk drive. Enter ^F to execute the command. The system will begin to store the information which is on the disk onto the hard disk. This will take several minutes. You will then be prompted to take out the original disk, and insert a new blank disk into the same disk drive. The blank disk will be automatically formatted and then the stored information will be placed on that disk. When the process is complete, you will have two identical disks.

SAVE/RESTORE MENU

INTRODUCTION

The Save/Restore Menu allows you to prepare floppy disks to be used for backups, perform the backup procedure (which is copying files onto the floppy disk or tape for safe keeping), and also restore the files back onto the hard disk system when they are needed.

```

mpadm-save      Save/Restore Operations 01/31/87 09:30

      Save Data                                Restore Data
A  Files and/or Directory                      R  Single File/Directory
S  Complete System                            Z  Complete System
      Prepare/Verify Media                      /  Change Directory

F  Formata Floppy Disk
C  Catalog Floppy Disk(s)

Quit-^X Help-^W More Keys-^Y Previous-^P

Select function by entering letter or using arrow keys and pressing
RETURN
  
```

/ - CHANGE DIRECTORY

From any menu you can change your position in the hierarchy of directories. The directory in which you are presently positioned is referred to as your "current directory". By simply pressing the "/" (slash) key, you can change your position from one directory to another.

When you press the "/", two lines appear at the bottom of the screen. The first is the name of your current directory (your current position).

Next appears a prompt line which reads:

Enter new directory name _____

Type in the name of the directory you wish to move to and press RETURN. You will then be moved to this new directory. If, however, the directory you specified does not exist, or was entered incorrectly, you will receive the message:

"Directory does not exist"
"Could not perform requested function. Press RETURN to continue or ^E to exit."

You can simply press RETURN to try again, or press ^E to abandon your attempt to change your current directory position.

Examples:

You are located in /usr/tmp. You move to directory /usr/acct by pressing the "/" key, and then typing "/usr/acct" and pressing RETURN.

Suppose further that after moving to /usr/acct, you wish to change to a sub-directory that is called /usr/acct/bob. All you need to enter is "bob".

To change back to the former directory, you must enter "/usr/acct".

A - [SAVE] FILES AND/OR DIRECTORIES

This command allows you to backup files or whole directories onto floppy disks or a tape. **MAKE SURE YOU HAVE ENOUGH DISKS FORMATTED BEFORE YOU BEGIN THIS PROCEDURE.** You will need at least one high density disk (which can hold 1.2 megabytes), or three standard DSDD 360K disks for every megabyte (1024 kbytes).

Name of device for backup ___
File/Directory name(s) to be backed up _____

First specify the device containing the floppy disk. You may press ^W (HELP) for a list of backup devices. Enter the two character name of the device in which the floppy disk has been placed. **WRITE DOWN ON A PIECE OF PAPER THE FULL NAME OF THE DEVICE YOU SELECTED**, such as "/dev/rdisk/fd096". Press ^W (HELP) again, when you are on the next line, to display the contents of the current directory.

If you want to backup the contents of the directory you are currently positioned in, and all of its subdirectories, enter a period "." on the line.

If you want to backup specific files, enter those names, with a space or a comma separating each name.

If you want to backup a directory, other than the directory you are in, enter the directory name, such as `"/tmp"`.

Enter `^F` to execute the command.

The backup procedure will begin. The file names are displayed on the screen as they are backed up. As each disk is filled up a message will appear on the screen. It will say:

errno 6 Can't write output If you want to go on, type device/file name when ready.

Take the disk out, number and label it, and load the next one. Type in the FULL DEVICE NAME THAT YOU WROTE DOWN. Press RETURN to continue. When finished, store the disks in a safe place.

S - [SAVE THE] COMPLETE SYSTEM

This command allows you to backup the complete hard disk system onto formatted floppy disks. MAKE SURE YOU HAVE ENOUGH DISKS FORMATTED BEFORE YOU BEGIN THIS PROCEDURE. You will need at least one high density disk (which can hold 1.2 megabytes), or three standard DSDD 360K disks for every megabyte (1024 kbytes).

Name of device for backup

Specify the device containing the floppy disk. You may press `^W` (HELP) for a list of backup devices. Enter the two character name of the device in which the floppy disk has been placed. WRITE DOWN ON A PIECE OF PAPER THE FULL NAME OF THE DEVICE YOU SELECTED, such as `"/dev/rdisk/fd096"`.

Make sure the floppy disk or tape is loaded correctly.

Enter `^F` to execute the command.

The backup procedure will begin. The file names are displayed on the screen as they are backed up. As each disk is filled up a message will appear on the screen. It will say:

errno 6 Can't write output If you want to go on, type device/file name when ready.

Take the disk out, number and label it, and load the next one. Type in the FULL DEVICE NAME THAT YOU WROTE DOWN. Press RETURN to continue. When finished, store the disks in a safe place.

F - FORMAT A FLOPPY DISK

This command allows you to format a floppy disk, which means to prepare it for use on your floppy disk drive.

Name of device containing floppy _____
to be formatted

First you must select the name of the device which contains the disk. You may press ^W (HELP) to display the list of devices available. Enter the two character NAME of the device you wish to use.

Make sure the floppy disk is loaded correctly.

Enter ^F to execute the command.

The screen will display the message that the formatting process is taking place, and on which device. When formatting is complete, you will be prompted to press RETURN to go back to the menu.

C - CATALOG FLOPPY DISK(S)

This command allows you to display a list of all the files stored on a floppy disk.

Name of device for cataloguing _____

You may press ^W (HELP) to display the list of devices available. Enter the two character NAME of the device in which the floppy has been placed.

Make sure the floppy disk or tape is loaded correctly.

Enter ^F to execute the command.

The information will appear on the screen. The file name is on the far right, next is the date it was created, then the file size in bytes, and then the ownership of the file. The numbers on the left are the file owner's identifiers. When the listing is complete you will be prompted to press RETURN to go back to the menu.

R - [RESTORE] SINGLE FILE/DIRECTORY

This command allows you to restore files or whole directories back to the hard disk after they have been backed up onto a floppy disk. You can indicate where you want the files to be restored, and choose which files you want restored.

Name of device for restore _____

Destination directory for files being restored (optional) _____

Names of files/directories to be restored (optional) _____

You may press ^W (HELP) to display the list of backup devices available. Enter the two character NAME of the device in which the floppy disk has been placed. **WRITE DOWN ON A PIECE OF PAPER THE FULL NAME OF THE DEVICE**, such as `"/dev/rdisk/fd096"`.

If you do not enter a destination directory for the files, they will be restored in the current directory that you are in. If you want the files to be restored in another directory, enter that name, such as `"/tmp"`. (If files were backed up using complete path names, then they will be restored at these same locations).

If you do not enter the name of files/directories to be restored, everything on the floppy disk will be restored to the destination that you specified. If you want only certain files or directories, enter those names.

A short cut to restoring certain files is the use of patterns for file name matching. The "*" (asterisk or star, located above the 8) is a "wild card". It will allow you to match many files by just specifying part of the file name. For example, you want to restore all files that end with the letters "prop" (for proposal). You had saved SMITHPROP, JONESPROP, and BROWNPROP. Now, instead of entering each file name, you can match the PROP, by specifying `"*PROP"`. All files that end in PROP will be restored. Here's another example; you could restore all files that begin with the letter C, by specifying `"C*"`.

Make sure that the floppy disk is loaded correctly.

Enter ^F to execute the command.

The screen displays the message that the restore is in process. The names of the files that are being restored are displayed along with the number of blocks (the space) that they occupy. As each disk is completed a message will appear on the screen. It will say:

Display information about all processes? _

Generate a summary listing? _

Generate a very detailed listing? _

These prompts require Y(es) or N(o) answers. (Leaving a line empty is the same as entering No). As you go down the screen, each Y answer is a request for more information. The first will display information about all processes rather than only those associated with the terminal at which you are working. The second prompt will cause this list to contain some detailed information about each process. The third prompt, will produce an even more detailed account of process activity.

Display process associated with terminal _____

Display selected process ids _____

Display selected processes for user _____

The next three prompts allow you to restrict this list to particular sets of processes. For each, you can press ^W (HELP) for a list of processes and users that are on the system.

You can specify that process information be restricted to those associated with a given terminal(s), user(s), or a specific process.

Press ^F to execute the command.

T - TERMINATE USER PROCESSES

This command allows the System Administrator to terminate/kill a process. It is often used to terminate a process which has caused a terminal to become locked (hung).

Process IDs _____

To help you determine the ID of the offending process press ^W (HELP) for the list of processes and users that are on the system.

Enter the Process ID number(s) that you want to terminate on the line. If there is more than one, separate the information by commas or spaces.

Press ^F to execute the command.

B - SEND BROADCAST MESSAGE

This command allows the System Administrator to send a message to all the users on the system.

Message to be sent _____

Enter the message you want to send on the line. Up to 8 lines are available.

If you wish to know who is logged on to the system press ^W (HELP) for a list of these users.

Press ^F to execute the command.

W - SEND MESSAGE TO A USER

This command allows you to send (write) a message directly to the screen of another user.

Enter the name of the person to
write to _____

Indicate which terminal to write to if
the user is logged in more than once _____

You can press ^W (Help) to list all users and the terminals they are currently using.

Next enter the login name of the message's intended recipient. If that user has logged in at more than one terminal, fill in the next line with the name of the terminal they are currently using.

Press ^F to execute the command.

R - DISPLAY RUN LEVEL

This command allows the System Administrator to display the run level of the system. The run level determines at which terminals users may log in. Refer to the Machine Management Menu for more about setting up terminals to run at specified run levels.

When you press R the screen will be cleared and the current run level of the system will be displayed. Press RETURN to then go back to the menu.

D - SET THE SYSTEM DATE

This command allows the Administrator to set the internal clock of the system to a specified date and time. You can press ^W (HELP) to display the current date and time.

- Enter the number of the current month _____
- Enter the day of the month _____
- Enter the hour in military time _____
- Enter the minutes _____
- Enter the year number (last 2 digits) _____

Fill in the blanks with the correct two digit information and press ^F to execute the command.

S - SINGLE USER MODE

This command allows the System Administrator to bring the system down to single user mode. In this mode only the system console is online.

- Grace period (seconds) before shutdown
to single user mode _____
- Are you sure you want to proceed _____

You can specify the number of seconds which this command will wait before entering single user mode. If no grace period is specified, a 60 second grace period will be observed.

You must answer Y(es), that you want to proceed, in order for this command to execute.

Press ^F to execute the command.

F - CHECK FILE SYSTEM FOR ERRORS

This command allows the System Administrator to check the `/usr` file system for errors. It is recommended that the system be in a single user state when this procedure is run. You must position yourself in the root directory (`/`) by using the Change Directory command (press the `/` key and respond with a slash (`/`) before you start this procedure). If you don't change to the root directory, the procedure will automatically terminate with an error message.

```

Check the /usr file system?          Y
(device /dev/dsk/0s2)

```

To check a file system, it must first be unmounted by the computer. If there are other users logged on and currently using the `/dev/dsk/0s2` device (or if you are not in the root directory), the command will terminate and display an error message. Otherwise, the screen will be cleared and the system will report its progress in checking the `/usr` file system.

You must answer Y(es), that you want to proceed, in order for this command to execute.

E Press `^F` to execute the command.

Z - SHUTDOWN THE SYSTEM

This command allows the System Administrator to shutdown the system.

```

Grace period (seconds) before shutdown  _
Are you sure you want to proceed        _

```

You can press `^W` (HELP) to list those users that are presently logged on to the system.

You can specify the number of seconds which this command will wait before shutting down the system. If no grace period is specified, a 60 second grace period will be observed.

You must answer Y(es), that you want to proceed, in order for the command to execute.

Press `^F` to execute the command.

P - PATCH KERNEL FOR TTY TERMINALS

The System Administrator can use this utility to modify the kernel to recognize additional TTY ports. The specific parameters are dependent on the communications board being installed as well as the number of boards. For more information refer to the TTYPATCH (1) command in the UNIX reference manual and the board manufacturer's documentation.

A - ADD A PRINTER

This command allows the System Administrator to add and/or configure a printer in the system.

```

Printer name                _____
Type of printer             _____
Type of printer (P=Parallel, S=Serial) _
Communication Speed
(serial connection only)    _____
Device to associate with printer _____
Set as system default destination _
    
```

Enter the name of the new printer to be added to the system. If necessary, press ^W (HELP) to display a list of printers already on the system and their status.

Enter printer type. If necessary, press ^W (HELP) to display a list of supported printer types. The default is "dumb" and it is recommended that you use this value. This associates a printer driver with the device. The model drivers are stored in the directory /usr/spool/lp/model.

Enter either P or S for parallel or serial printer. The default is "P" (for parallel).

Enter the communication speed. If necessary, press ^W (Help) for a list of available speed codes.

Enter the name of the device the printer is connected to. Press ^W (HELP) to display a list of devices that are available. The default is "/dev/lp" which is usually connected to the first parallel port. Serial ports are designated tty0 through tty16. The number of serial ports available on your system is determined by the number of communications boards installed. Most systems usually are configured with one or two ports as standard.

Answer with a Y or N to set as the system default destination. The default is "N".

This command will configure the printer and then instruct it to begin accepting requests and then enable it so it may begin printing immediately. The printer scheduler is stopped at the start of configuration and restarted when the command has completed.

Enter ^F to execute the command.

If you wish to administer more sophisticated kinds of printer configurations, construct custom interface programs or group several printers into a class, please refer to the LP Spooling system description in your System V/AT Runtime System manual.

R - REMOVE A PRINTER

This command allows the System Administrator to remove a printer from the system. The command will automatically send the necessary PRINTER CONTROL information to the system.

Name of printer to be deleted _____

Enter the name of the printer that is to be removed as a destination. If necessary, press ^W (Help) to display a list of printers and their status.

For the system to delete a printer, it is necessary for it to temporarily shut down the print scheduler. When it is complete, the scheduler will be restarted automatically.

Enter ^F to execute the command.

X - SET SYSTEM DEFAULT

This command allows the System Administrator to set a new system default printer/destination. This will route all user output that does not specify a printer to this location. The default location is sometimes referred to as the system printer.

Name of new system default destination (printer) _____

SYSVISION

Enter the name of the printer that is to be the new system default printer. If necessary, press ^W (HELP) to display a list of printers and their status.

Enter ^F to execute the command.

U - UUCP SETUP MENU

This menu allows the System Administrator to manage system and communication device definitions. Enter U and the UUCP Setup Menu will be displayed.

U - UUCP SETUP MENU

INTRODUCTION

This menu allows the System Administrator to set up UNIX to UNIX (UUCP) communications by defining the entries for remote systems and communication devices.

```

mpadmuucp  UUCP Configuration 01/31/87 09:30

      Systems                               Devices
S  Add a System                             R  Add a Device
D  Delete a System                          X  Delete a Device

C  Call a System

N  Set UUCP Nodename

Quit-^X Help-^W More Keys-^Y Previous-^P

Select function by entering letter or use arrow keys and pressing RETURN

```

Select the letter or character for the function you wish to perform and proceed to the corresponding command in this section.

S - ADD A SYSTEM

This command allows a new system definition to be added to the system. This adds an entry to the L.sys file which is located in the /usr/lib/uucp directory. Much of the information that is entered into this screen must be provided to you by the system administrator of the remote system that you are defining.

SYSVISION

System Name _____

Description of System _____

Connection type to remote system _____

Communication line speed setting _____

Phone number of remote system _____
(If direct line, leave blank)

Name of communication port to use for directly connected systems (if dialup, leave blank) _____

Enter user account name for UUCP _____

Enter password for account name _____

Enter the System Name. Press ^W (Help) for a list of known systems with the NAME and DESCRIPTION to see what systems are already available. This must be the name by which the remote system identifies itself when dialed up.

Enter the Description of the system. This can be the name of the system or owner of the system.

Enter the connection type. This will be either a "1" for a dialup link, or "2" for a direct connection.

Enter the communication line speed. You can press ^W (Help) for a list of communication ports and their associated speeds.

If the system is connected via a dialup link, enter the phone number. If not, leave blank.

If the system is directly connected, enter the name of the port that will be used. Press ^W (Help) for the list of ports available.

For example:

Connect	Port	Modem Type	Speed
ACU	tty0	hayes	1200
DIR	tty1	direct	9600

In this example, port "tty1" would be entered. The first column indicates what type of connection is defined for this port. ACU (Automatic Call Unit) usually means that an autodial modem is attached to the port. DIR signifies that a direct connection to another system is attached to this port.

Enter the user account name for UUCP. This information should be provided to you by the System Administrator of the system you are configuring.

Enter the password for the Account name. This information should be provided to you by the System Administrator of the system you are configuring.

Enter ^F to execute the function.

D - DELETE A SYSTEM

This command allows you to delete a system definition from the system.

System to be deleted _____

Enter the System Name to be deleted. If necessary, press ^W (Help) to display a list of systems available for deletion.

Enter ^F to execute the command.

C - LOG ONTO ANOTHER SYSTEM

This command allows you to call and log onto another computer system. You can use your terminal as a workstation for another computer, not just the one you normally use.

System to be called _____

Enter the name of the system you wish to access. If you don't know the name of the system press ^W (Help) for a list of computer systems that are available for you to log onto. A list of system names and associated descriptions will be displayed in the help window.

Once you have entered the appropriate system name, press ^F to execute the command.

After you have finished using the remote system, be sure to logout and then terminate the session by typing ~. followed by a carriage return, which will instruct your system to break off communications with the remote system. For more information, please refer to the cu(1) command in the System V/AT Runtime System Manual.

N - SET UUCP NODE NAME

This command allows the System Administrator to change the UUCP (UNIX to UNIX Copy) Node name of his system.

Enter the new name of your system _____

Press ^W (Help) to display the current system name and a list of other known systems. Enter the new name of your system. It cannot duplicate the name of any of the other known systems.

Enter ^F to execute the command.

R - ADD A DEVICE

This command allows you to add a new communication device to the system. This adds an entry to the L-devices file which is located in the /usr/lib/uucp directory.

Communication device type _____
(defaults to 1)

Port name _____

Enter Modem Type _____

Communication line speed setting _____

Enter the communication device type. The connection type will be either a "1" for a dialup link, or "2" for a direct connection. The default is 1.

Enter the port name. If necessary, press ^W (Help) for a list of available ports.

Enter the modem type. If necessary, press ^W (Help) for a list of available modem types. For example:

Name	Description
direct	Direct communication link (no modem)
uhayes	Unix Hayes
hayes	Hayes
Vadic	Racal Vadic 3451

Enter the communication line speed setting. The allowed speeds are 300, 1200, 2400, and 9600 baud. This is dependent on the type of device or system attached to the port. Check the modem documentation or consult with the system administrator of the system you are connecting.

Enter ^F to execute the command

X - DELETE A DEVICE

This command allows you to delete a communications device from the system.

Port name _____

Enter the port name of the device to be deleted. If necessary, press ^W (Help) for a list of devices and their ports.

For example:

Connect	Port	Modem Type	Speed
ACU	tty0	hayes	1200
DIR	tty1	direct	9600

Enter ^F to execute the command.

Chapter 3

WHAT IS THE UNIX SYSTEM?

WHAT THE UNIX SYSTEM IS

C-3

The UNIX system is a set of programs, called software, that acts as the link between a computer and you, its user. The UNIX system is designed to control the computer on which it is running so the computer can operate efficiently and smoothly and to provide you with an uncomplicated, efficient, and flexible computing environment.

UNIX system software does three things:

- It controls the computer,
- It acts as an interpreter between you and the computer, and
- It provides a package of programs or tools that allows you to do your work.

The UNIX system software that controls the computer is referred to as the operating system. The operating system coordinates all the details of the computer's internals, such as allocating system resources and making the computer available for general purposes. The nucleus of this operating system is called the kernel.

In the UNIX system, the software that acts as a liaison between you and the computer is called the shell. The shell interprets your requests and, if valid, retrieves programs from the computer's memory and executes them.

The UNIX system software that allows you to do your work includes programs and packages of programs called tools for electronic communication, for creating and changing text, and for writing programs and developing software tools.

WHAT IS THE UNIX SYSTEM?

Put simply, this package of services and utilities called the UNIX system offers:

- A *general purpose* system that makes the resources and capabilities of the computer available to you for performing a wide variety of jobs or applications, not simply one or a few specific tasks.
- A computing environment that allows for an *interactive* method of operation so you can directly communicate with the computer and receive an immediate response to your request or message.
- A technique for sharing what the system has to offer with other users, even though you have the impression that the UNIX system is giving you its undivided attention. This is called *timesharing*. The UNIX system creates this feeling by allowing you and other users--*multiusers*--slots of computing time measured in fractions of seconds. The rapidity and effectiveness with which the UNIX system switches from working with you to working with other users makes it appear that the system is working with all users simultaneously.
- A system that provides you with the capability of executing more than one program simultaneously, this feature is called *multitasking*.

The UNIX system, like other operating systems, gives the computer on which it runs a certain profile and distinguishing capabilities. But unlike other operating systems, it is largely machine-independent; this means that the UNIX system can run on mainframe computers as well as microcomputers and minicomputers.

From your point of view, regardless of the size or type of computer you are using, your computing environment will be the same. In fact, the integrity of the computing environment offered by the UNIX system remains intact, even with the addition of optional UNIX system software packages that enhance your computing capabilities.

HOW THE UNIX SYSTEM WORKS

After reading the past few pages, you know that the UNIX system offers you a set of software that performs services--some automatically, some you must request. You also know that the system creates a certain environment in which you can use its software. But before you can ask the UNIX system to do something, you need to know what it is capable of doing.

Look at *Figure 1-1*. It shows a set of layered circles in graduated sizes. Each circle represents specific UNIX system software, such as:

- Kernel,
- Shell, and
- Programs/tools that run on command.

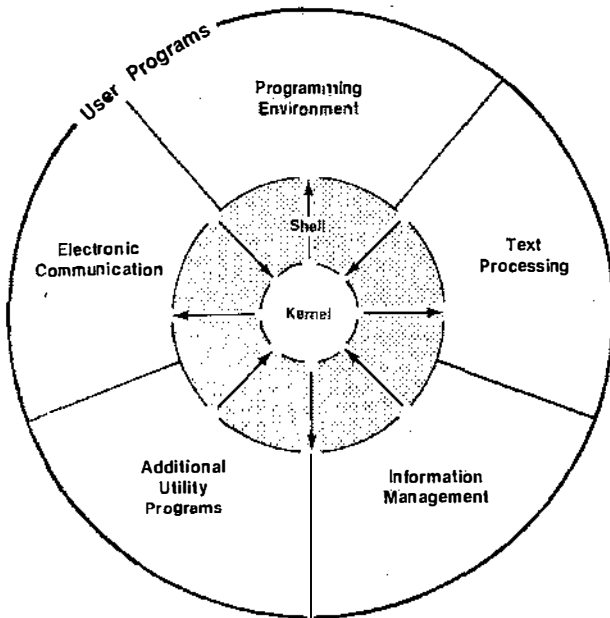


Figure 1-1. UNIX system model

WHAT IS THE UNIX SYSTEM?

You should know something about the major components of UNIX system software to communicate with the UNIX system. Therefore, the remainder of this chapter introduces you to each component: the kernel, the shell, and user programs or commands.

Kernel

The heart of the UNIX system is called the kernel. *Figure 1-2* gives an overview of the kernel's activities. Essentially, the kernel is software that controls access to the computer, manages the computer's memory, and allocates the computer's resources to one user, then to another. From your point of view, the kernel performs these tasks automatically. The details of how the kernel accomplishes this are hidden from you. This arrangement lets you focus on your work, not on the computer's.

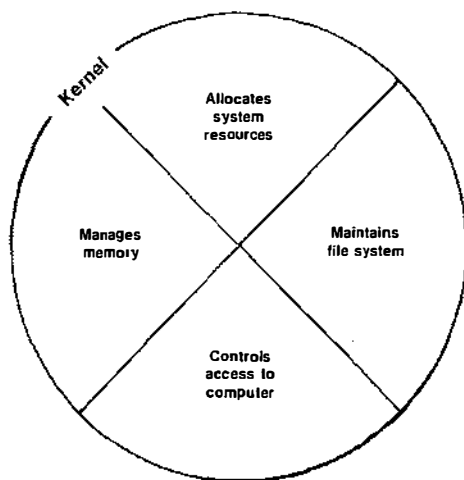


Figure 1-2. Functional view of kernel

On the other hand, you will become increasingly familiar with another feature of the kernel; this feature is referred to as the file system.

The file system is the cornerstone of the UNIX operating system. It provides you with a logical, straightforward way to organize, retrieve, and manage information electronically. If it were possible to see this file system, it might look like an inverted tree or organization chart made up of various types of files *Figure 1-3*. The file is the basic unit of the UNIX system and it can be any one of three types:

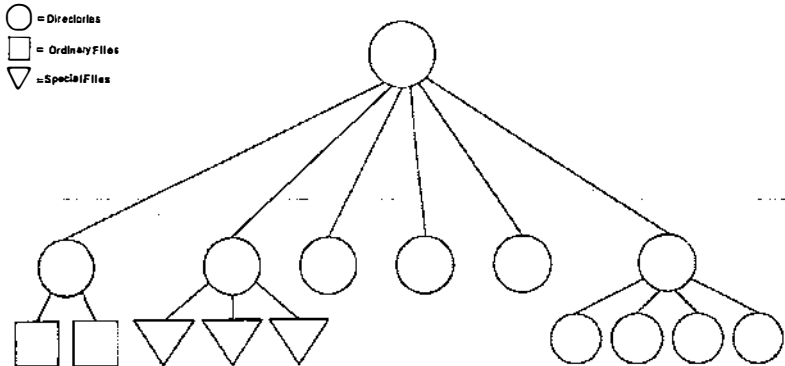


Figure 1-3. Branching directories and files give the UNIX system its treelike structure

- An *ordinary file* is simply a collection of characters. Ordinary files are used to store information. They may contain text or data for the letters or reports you type, code for the programs you write, or commands to run your programs. In the UNIX system, everything you wish to save must be written into a file.

In other words, a file is a place for you to put information for safekeeping until you need to recall or use its contents again. You can add material to or delete material from a file once you have created it, or you can remove it entirely when the file is no longer needed.

WHAT IS THE UNIX SYSTEM?

- A *directory* is a file maintained by the operating system for organizing the treelike structure of the file system. A directory contains files and other directories as designated by you. You can build a directory to hold or organize your files on the basis of some similarity or criterion, such as subject or type.

For example, a directory might hold files containing memos and reports you write pertaining to a specific project or client. Or a directory might hold files containing research specifications and programming source code for product development. A directory might hold files of executable code allowing you to run your computing jobs. Or a directory might contain files representing any combination of these possibilities.

- A *special file* represents a physical device, such as the terminal on which you do your computing work or a disk on which ordinary files are stored. At least one special file corresponds to each physical device supported by the UNIX system.

In some operating systems, you must define the kind of file you will be working with and then use it in a specified way. You must consider how the files are stored since they can be sequential, random-access, or binary files. To the UNIX system, however, all files are alike. This makes the UNIX system file structure easy to use. For example, you need not specify memory requirements for your files since the system automatically does this for you. Or if you or a program you write needs to access a certain device, such as a printer, you specify the device just as you would another one of your files. In the UNIX system, there is only one interface for all input from you and output to you; this simplifies your interaction with the system.

The source of the UNIX system file structure is a directory known as root, which is designated with a slash (/). All files and directories in the file system are arranged in a hierarchy under root. Root normally contains the kernel as well as links to several important system directories that are shown in *Figure 1-4*:

/bin	Many executable programs and utilities reside in this directory.
/dev	This directory contains special files that represent peripheral devices, such as the console, the line printer, user terminals, and disks.

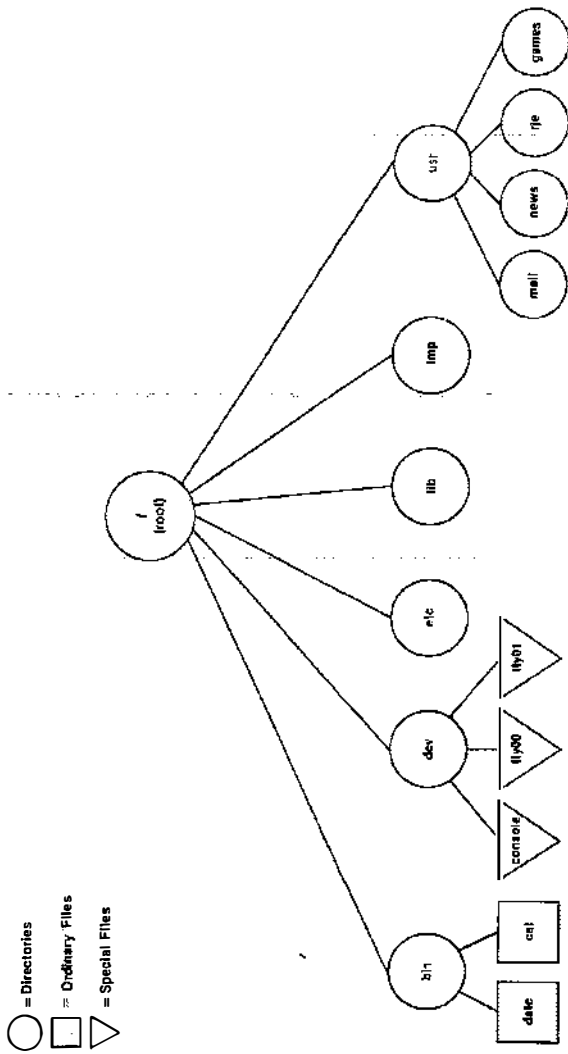


Figure 1-4. Sample of typical file system structure

WHAT IS THE UNIX SYSTEM?

- /etc** Programs and data files for system administration can be found in this directory.
- /lib** This directory contains available program and language libraries.
- /tmp** This directory is a place where anyone can create temporary files.
- /usr** This directory holds other directories, such as **mail** (which further holds files storing electronic mail), **news** (which contains files holding newsworthy items), **rje** (which contains files needed to send data via something called the remote job entry communication link), and **games** (which contains files holding electronic games).

In summary, the directories and files you create comprise the portion of the file system that is structured and, for the most part, controlled by you. Other parts of the file system are provided and maintained by the operating system, such as **bin**, **dev**, **etc**, **lib**, **tmp** and **usr**, and have much the same structure on all UNIX systems.

The "Using the File System" chapter of this manual shows how to organize a file system directory structure and how to access and manipulate files. "Unix System Capabilities" gives an overview of UNIX system capabilities. The effective use of these capabilities depends on your familiarity with the file system and your ability to access information stored within it. The "Screen Editor Tutorial" is designed to teach you how to create and edit files to meet your computing and information management needs.

Shell

The shell is a unique UNIX system program or tool that is central to most of your interactions with the UNIX system. *Figure 1-1* illustrates how the shell works. The drawing shows the shell as a circle containing arrows pointing away from the kernel and the file system to the outer circle that contains programs and then back again. The arrows indicate that a two-way flow of communication is possible between you and the computer via the shell.

When you enter a request to the UNIX system by typing on the terminal keyboard, the shell translates your request into language the computer understands. If your request is valid, the computer honors it and carries out an instruction or set of instructions. Because of its job as translator, the shell is called the command language interpreter.

As the command language interpreter, the shell can also help you to manage information. The shell's ability to manage information stems from the design of the UNIX system. Each program in the UNIX system is designed to do one thing well. In a sense, a UNIX system program is a building block or module that you can use in tandem with other programs to create even more powerful tools.

In addition to acting as a command language interpreter, the shell is a programming language complete with variables and control flow capabilities.

A section of the next chapter describes each of the shell's capabilities. Any reference work on shell programming techniques can teach you how to use these capabilities to write simple shell programs called shell scripts and how to custom-tailor your computing environment.

Commands

A program is a set of instructions that the computer follows to do a specific job. In the UNIX system, programs that can be executed by the computer without need for translation are called executable programs or commands.

As a typical user of the UNIX system, you have many standard programs and tools available to you. If you also use the UNIX system to write programs and to design and develop software, you have system calls, subroutines, and other tools at your disposal. And you have, of course, the programs you write.

This book introduces you to approximately 40 of the most frequently used programs and tools that you will probably use on a regular basis when you interact with the UNIX system. If you need additional information on these or other standard UNIX system programs, check section 1 of the Runtime System manual.

WHAT IS THE UNIX SYSTEM?

If you want to use tools and routines that relate to programming and software development, you should consult the Software Development System manual.

The details contained in the two reference manuals may also be available via your terminal in what is called the *on-line* version of the UNIX system reference manuals. This on-line version is made up of formatted text files that look exactly like the printed pages in the manuals. You can summon pages in this electronic manual using the command **man**, which stands for **man**ual **pa**ge, if the electronic version of the manuals is available on your computer. The **man** command is documented in your copy of the Runtime System manual.

What Commands Do

The outer circle of *Figure 1-1* organizes UNIX system programs and tools into general categories according to what they do. The programs and tools allow you to:

- *Process text.* This capability includes programs, such as, line and screen editors (which create and change text), a spelling checker (which locates spelling errors), and optional text formatters (which produce high-quality paper copies that are suitable for publication).
- *Manage information.* The UNIX system provides many programs that allow you to create, organize, and remove files and directories.
- *Communicate electronically.* Several programs, such as **mail**, provide you with the capability to transmit information to other users and to other UNIX systems.
- *Use a productive programming and software development environment.* A number of UNIX system programs establish a friendly programming environment by providing UNIX-to-programming-language interfaces and by supplying numerous utility programs.
- *Take advantage of additional system capabilities.* These programs include graphics, a desk calculator package, and computer games.

How Commands Execute

Figure 1-5 gives a general idea of what happens when the UNIX system executes a command.

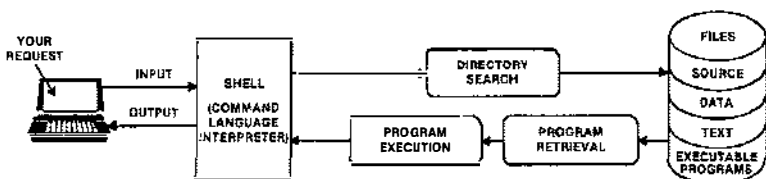


Figure 1-5. Flow of control between you and computer when you request program to run

When the shell signals it is ready to accept your request, you type in the command you wish to execute on the keyboard. The command is considered input, and the shell searches one or more directories to locate the program you specified. When the program is found, the shell brings your request to the attention of the kernel. The kernel then follows the program's instructions and executes your request. After the program runs, the shell asks you for more information or tells you it is ready for your next command.

This is how the UNIX system works when your request is in a format that the shell understands. The structure that the shell understands is called a command line. "Using the File System" explains what you need to know about the command line so you can request a program to run.

This chapter has outlined some basic principles of the UNIX operating system and explained how they work. The following chapters will help you begin to apply these principles according to your computing needs.

10

11

12

Chapter 4

UNIX SYSTEM CAPABILITIES

INTRODUCTION

This chapter serves as a transition between the first three chapters in the overview part of this guide and the four tutorials that follow. The material in this chapter combines basic, fundamental concepts about the UNIX system covered in the first three chapters of this guide with information about system capabilities that you may use to do your computing work efficiently and effectively.

This chapter provides an overview of the following UNIX system capabilities: text editing, working in the shell, communicating electronically, and programming in the UNIX system environment. In addition, it serves as an introduction to the "Screen Editor Tutorial" and "Communication Tutorial" chapters.

TEXT EDITING

You have read a good deal about files up to this point simply because using the file system is a way of life in a UNIX system environment. The information in this section will enhance your knowledge about manipulating files by introducing you to a software tool called a text editor. A text editor provides you with the ability to create and modify files: it will help you to fare well in the UNIX system since a considerable amount of your computing time may be spent writing and revising letters, memos, reports, or source code for programs that will be stored in files.

This section contains information that tells you what a text editor is and how it works. In addition, this section acquaints you with two types of text editors supported on the UNIX system: the line editor and the visual, or screen, editor. Since you will probably come to prefer one of these editing programs over the other--even if you learn to use them equally well--the line editor and the screen editor

are briefly compared to help you to assess their capabilities. For detailed information on the line editor and the screen editor, see "Screen Editor Tutorial."

What Is a Text Editor?

When you write or type letters, memos, and reports and then decide to change what you have written or typed, you will use skills required in text editing. These skills include inserting new or additional material, deleting unneeded material, transposing material (sometimes called cutting and pasting), and finally preparing a clean, corrected copy. Text editors perform these tasks at your direction making writing and revising text much easier and quicker than if done by hand or on a typewriter.

In the UNIX system, a text editor is much like the UNIX system shell. Both a text editor and the shell are programs that accept your commands and then perform the requested functions--essentially, they are both interactive programs. A major difference between a text editor and the shell, however, is the set of commands that each recognizes. All the commands you have learned up to this point belong to the shell's command set. A text editor, on the other hand, has its own distinct set of commands that allow you to create, move, add, and delete text in files, as well as acquire text from other files.

How Does a Text Editor Work?

To understand how a text editor works you need information about the environment created when you use an editing program and the modes of operation understood by a text editor.

Text Editing Buffers

To create a new file, you must ask the shell to put the editor in control of your computing session. When you do, a temporary work space is allocated to you by the editor. This work space is called the editing buffer, in it you can enter information you want the file to hold and modify it if you wish.

Because you are in a temporary work space when using a text editor, the file you are creating along with the changes you make to it are also temporary. This work space allotment and what it is holding

will exist only as long as you work in the editing program. If you wish to save the file, you must tell the text editor to write the contents of the buffer into a storage area. If you do not tell the editor to write or record what you have done during the editing session, the buffer's contents will disappear when you leave the editing program. If you forget to write a new file or update an existing one, the text editors remind you to do so when you attempt to leave the editing environment.

To modify an existing file, the procedure is almost identical to the one you follow when creating a new file. First, call the editor and give it the name of the file you wish to change. In turn, the editor makes a copy of the file that is in the storage area and places it in the buffer so you can work on it.

When you finish editing the file, you can write the buffer's contents into storage and leave the editing program knowing the file is updated and ready to be recalled when you need it again. Or you can choose to leave the editor without writing the file if you have made a critical mistake or you are unhappy with the edited version. This step leaves the original file intact and the edited copy disappears.

Regardless of whether you are creating a new file or updating an existing one, the text you put in the buffer is organized into lines. A line of text is simply the series of characters that appears horizontally across a row of typing that is ended by pressing the <CR> key. Occasionally, files may contain a line of text that is too long to fit on the terminal monitor. Some terminals will automatically display the continuation of the line on the next row of the monitor, whereas others will not.

Modes of Operation

Text editors are capable of understanding two modes of operation: the command mode and the text input mode.

When you begin an editing session, you will automatically be placed in command mode. In command mode, all your input is interpreted as a command. Typical editing commands allow you to move about in a file, search for patterns in the file's contents, or print a portion of a file on the terminal monitor. The input mode is entered when you

use a command to create text. Once in input mode, what you type on the keyboard is placed into the buffer as part of the text file until you send the appropriate instruction to the editor that returns you to command mode.

You may occasionally lose track of the mode in which you are working by attempting to enter text while in command mode or by trying to enter a command while in input mode. This is something even experienced users do from time to time. It will not take long to recognize the mistake and it will be apparent what to do to remedy these situations as you work through the "Screen Editor Tutorial" chapter of this manual.

Line Editor

The line editor, accessed by the **ed** command, is a fast, versatile program for preparing text files. This editor gets its name because it operates on the lines of text a file holds. For example, to change a single character in a file, you specify the line of the file that contains the character you wish to change and then specify the change.

Put simply, you manipulate text on a line-by-line basis with the line editor. Commands for this text editor can change lines, print lines, read and write files, and initiate text entry. In addition, you can specify the line editor to run from a shell program; something you cannot do with the screen editor. (See an outside reference on UNIX shell programming for information on basic shell programming techniques.)

The line editor works equally well on paper printing terminals and video display terminals. It will also obligingly accommodate you if you are using a slow-speed telephone line.

If you are interested in a comparison of line editor (**ed**) and screen editor (**vi**) features, see *Table 4-1*.

TABLE 4-1
Comparison of Line (*ed*) and Screen (*vi*) Editors

Feature	Line Editor (<i>ed</i>)	Screen Editor (<i>vi</i>)
Recommended terminal type	Paper-printing or VDT*	VDT
Speed	Accommodates high- and low-speed data transmission lines.	Works best via high-speed data transmission lines (1,200+ baud).
Versatility	Can be specified to run from shell scripts as well as used during editing sessions.	Must be used interactively during editing sessions.
Sophistication	Changes text quickly. Uses comparatively small amounts of processing time.	Changes text easily. However, can make heavy demands on computer resources.
Power	Provides a full set of editing commands. Standard UNIX system text editor.	Provides its own editing commands and recognizes all line editor commands as well.

* VDT = video display terminal

Screen Editor

The screen editor, accessed by the *vi* command, is a display-oriented, interactive software tool. When you use the screen editor, your terminal acts as a window to let you view the file you are editing a screenful or page at a time. This editor works most efficiently and effectively when used on a video display terminal operating at 1,200 or higher baud.

For the most part, modifications to a file (such as, additions, deletions, and changes) are accomplished by positioning the cursor at the point in the window where the modification is to be made and then making the change. In other words, the screen editor displays the effects of editing changes in the context in which you make them.

UNIX SYSTEM CAPABILITIES

Because of this feature, the screen editor is considered to be much more sophisticated than the line editor.

Furthermore, the screen editor offers a replete collection of commands. For example, a number of screen editor commands allow you to move the cursor around within the window to a file. Other commands move the window up or down through a page or more of the file. Still other commands allow you to change existing text or to create new text. In addition to its own set of commands, the screen editor has access to all the commands offered by the line editor. This arsenal of commands accounts for the screen editor's tremendous power.

There is, however, a trade-off for the screen editor's speed, visual appeal, efficiency, and power, which is the heavy demand that it places on the computer's processing time. For example, a simple change might cause an entire screen to need updating. Moreover, if simple changes lead to long delays while you wait for a screen to be updated, the pleasant experience of using a visual-oriented editor can be somewhat diminished.

Refer to the "Screen Editor Tutorial chapter" for instructions on how to use this software. And see `vi(1)`, which contains a summary of screen editor commands. If you wish to compare the features of the line editor (`ed`) and the screen editor (`vi`) see *Table 4-1* of this chapter.

WORKING IN THE SHELL

Every time you log into the UNIX system you will be communicating directly with a program called the shell. You will continue to interact with the shell until you log off the system, unless you use a program, such as a text editor, that temporarily suspends your dealings with the shell until you are finished using that particular program.

The shell is much like other programs, except that instead of performing one job, as `cat` or `ls` does, it is central to most of your interactions with the UNIX system. This is because the shell's primary function is to act as an interpreter between you and the computer on which the UNIX system is running. As an interpreter,

the shell translates your requests into language the computer understands, calls requested programs into memory, and executes them.

This section acquaints you with some of the ways you can use the shell as the command language interpreter to simplify a computing session and to enhance your ability to use system features. In addition to running a single program for you, you can also use the shell to:

- interpret the name of a file or a directory you input in an abbreviated way using a type of "shell shorthand,"
- redirect the flow of input and output of the programs you run,
- execute multiple programs, and
- tailor your computing environment to meet your individual needs and preferences.

In addition to being the command language interpreter, the shell is also a programming language. If you would like an overview of shell programming capabilities, see the section entitled *Programming in the System* at the end of this chapter. Or refer to an outside reference for detailed information on how to use the shell as a command language interpreter and as a programming language. A separate document, *UNIX System Shell Commands and Programming*, should be consulted for complete, unabridged information on shell programming.

Using Shell Shorthand

Many UNIX system commands require that you name a file or a directory as an argument to it on a command line, such as `mkdir directory name(s)<CR>` or `rm filename(s)<CR>`. Easy enough! But suppose you have 12 files to remove corresponding to monthly reports for 1983 named *rept1*, *rept2*, *rept3*, *rept4*, and so on? Or suppose you need to move 24 files corresponding to file names *sect1*, *sect2*, ... *sect24* to a different directory?

UNIX SYSTEM CAPABILITIES

Typing the file name for each monthly report after the `rm` command or the file name for each section after the `mv` command is still easy, but all the repetition gets tedious after inputting four or five names.

In instances like these, you should consider using shorthand notation when specifying file or directory names. If the file or directory names have some part in common, you can use a type of shorthand to tell the shell that you are referring to all of them on the basis of the similarity without specifying each one individually. Or, if a file has a unique character or sequence of characters within a group of similarly named files, you can use this shorthand notation to locate the file on the basis of the difference.

The UNIX system recognizes several characters as having special meanings when they are used in place of a directory name or when they appear as part of a file or directory name on a command line. These characters allow you to specify the names of files and directories in a rapid, abbreviated way. Some of the characters are referred to as metacharacters because of their special meanings to the shell.

The special characters are `.` `..` `?` `*` `[` `]` `-` `\` and their meanings are summarized in *Table 4-2*. When you specify file or directory names, you can substitute various characters within them with the appropriate shorthand abbreviation. Any part of the name that is not a special character is taken at its literal value.

For example, for the possibilities described at the beginning of this section, typing `rm rept* <CR>` would remove all the files in the current directory starting with the characters *rept* followed by any other characters corresponding to monthly reports for 1983, and typing `mv sect* ../chapter3 <CR>` would move all the files from the current directory beginning with the letters *sect* and followed by any other characters to another directory named *chapter3* belonging to its parent directory.

Details on how to use the special characters `.` and `..` appear in the chapter called "Using the File System." The other special characters are called "shell metacharacters" and more detailed information on their use can be found in an outside reference work on UNIX shell programming.

TABLE 4-2
Shorthand Notation for File and Directory Names

Special Character	Meaning
.	Current directory
..	Parent directory
?	Match any single character
*	Match any number of characters
[]	Designate a sequence of characters to be matched, such as [abc] or [628]
-	Specify a character range within [], such as A-Z
\	Remove meaning of special characters

C-4

Redirecting the Flow of Input and Output

Up to this point in this chapter, any request to ask the shell to execute a command was done by inputting the command and the necessary argument(s) on the terminal keyboard. In turn, the output, if any, was displayed on the terminal monitor. This pattern illustrates the idea of standard input and standard output.

In general, the place from which a program expects to receive its input is called the standard input. A UNIX system command called **mail**, which you will learn more about in the "Communication Tutorial" chapter, provides a good example of this and warrants mentioning here. For example, to use **mail**, you would simply type **mail jmr**s <CR> and the **mail** command takes everything you type on your keyboard after <CR> until you type <^d> as input. After you type <^d>, mail sends your input to the person with the login name *jmr*s. The place to which a program writes its results, in this case the login name *jmr*s, is referred to as the standard output.

In the UNIX system, most commands expect to receive their input from the keyboard and then display output on the terminal monitor.

By default, then, the standard input is the keyboard and the standard output is the terminal monitor (*Figure 4-1*).

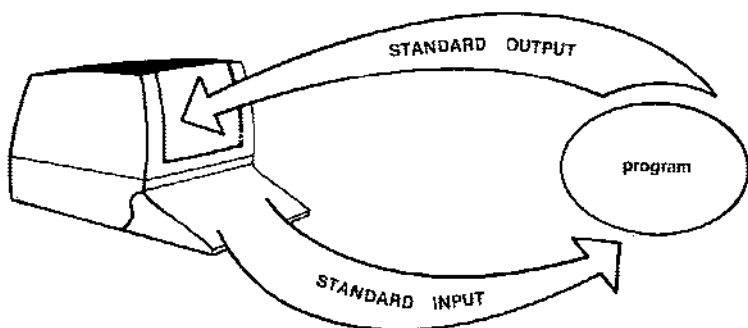


Figure 4-1. Standard input/output flow. A program's standard input and standard output are usually assigned to your terminal.

You can, if you wish, use a feature called redirection to change these defaults. Put simply, redirection is a UNIX system feature that allows you to request the shell to reassign standard input and/or standard output to other files or devices.

With the redirection feature, you can request the shell to do the following:

- reassign to a file any output that a program would ordinarily send to your terminal,
- have a program take its input from a file rather than from your terminal keyboard, or
- use a program as the source of data for another program.

You request the shell to redirect input and output using a set of operators, which are `>` (greater than sign), `>>` (two greater than signs), `<` (less than sign), and `|` (a pipe). Now let's take a look at what each of these operators can do for you.

Redirecting the Standard Output (>)

The > operator allows you to redirect the output of a command (or program) into a file (Figure 4-2).

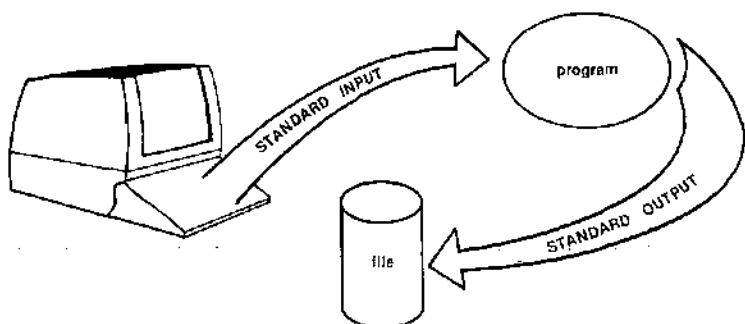


Figure 4-2. Standard output can be redirected from your terminal to a file.

To use the > operator, follow the command line format:

```
command > newfile <CR>
```

in which you can choose to surround the > operator with spaces as indicated in the command line or leave the spaces out (**command>newfile<CR>**); either method is correct.

For example, if you have two files, named *group1* and *group2* each containing a list of names with telephone extension numbers that you would like to sort alphabetically and then interfile into a separate file called *members*, you would type:

```
sort group1 group2 > members <CR>
```

When you do, the UNIX system first alphabetically sorts and then interfiles the contents of the files *group1* and *group2* and redirects the

UNIX SYSTEM CAPABILITIES

output into the file called *members* rather than displaying it on your terminal. If you wish to read the contents of the *members* file, you could use the *cat* or *pg* command.

Therefore, if the contents of the file *group1* is:

Smith, Allyn	101
Jones, Barbara	203
Cook, Karen	521
Moore, Peter	180
Wolf, Robert	125

and the contents of the file *group2* is:

Frank, M. Jay	118
Nelson, James	210
West, Donna	333
Hill, Charles	256
Morgan, Kristine	175

then the file *members* would appear as follows on your terminal when displayed with the *cat* command.

```
$ sort phase1 phase2 > members<CR>
$ cat members<CR>
Cook, Karen          521
Frank, M. Jay        118
Hill, Charles        256
Jones, Barbara       203
Moore, Peter         180
Morgan, Kristine     175
Nelson, James        210
Smith, Allyn         101
West, Donna          333
Wolf, Robert         125
$
```

Keep in mind that if the file to which you are redirecting the standard output already exists, its contents will be replaced with the output of the redirection command.

Redirecting and Appending the Standard Output (>>)

Occasionally, you might like to add information to the end of an existing file. You can use the >> operator to do so. Simply input the following command line:

```
command >> file<CR>
```

For example, if the file *members* that was created in the previous section was subject to additions and deletions, it might be a good idea to date the list so you know at a glance what version of the list you are using. You could do so by typing

```
date >> members<CR>
```

on the command line and the date and time would be printed at the end of the file *members*. Or instead of adding the date to the end of the file *members*, you could have appended another file containing even more names.

Redirecting the Standard Input (<)

Standard input can be redirected as well as standard output with the < operator. The general command line format for input redirection is:

```
command < file<CR>
```

in which the < operator informs the command (or program) to take input from the file you specify rather than from the terminal keyboard (Figure 4-3).

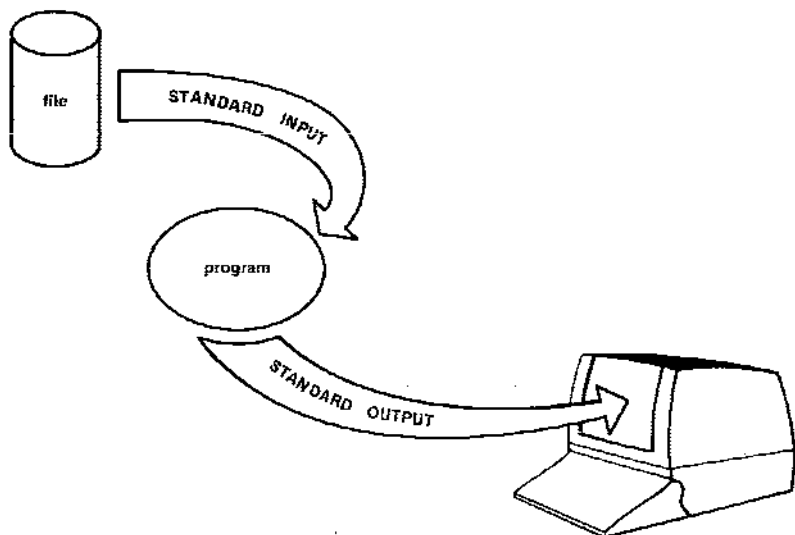


Figure 4-3. You can ask the shell to take a program's input from a file rather than from your terminal.

For example, if you would like to send a copy of the file *members* to co-workers who work on your UNIX system and who have the login names *mary2* and *jmrs*, typing

```
mail mary2 jmrs < members<CR>
```

will accomplish the task. The `mail` command, however, does not know whether it received its input from the file *members* (which it did) or from your keyboard. Rather, input/output redirection is a service provided by the UNIX system shell and is available to every program. (You will learn more about the `mail` command in the "Communications Tutorial" chapter.)

Connecting Commands with the Pipe (|)

The pipe operator is a powerful, yet flexible, mechanism for doing computing tasks quickly and without the need to develop special

purpose tools. You can use it to redirect the standard output of one program to be the standard input of another (Figure 4-4). Generally, the format for using the pipe is:

```
command | command <CR>
```

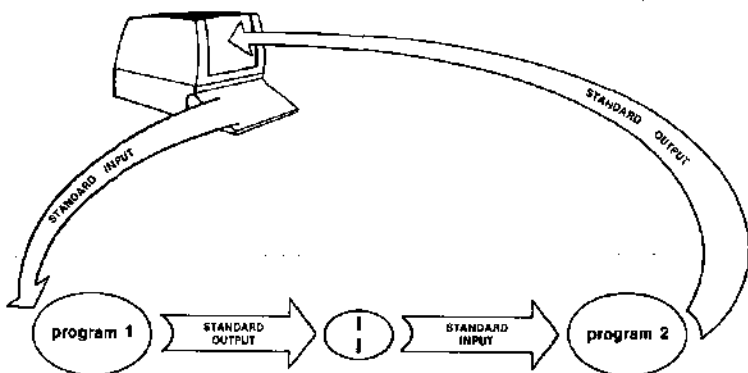


Figure 4-4. You can use the output from one program to be the input for another.

A popular example of this is taking the output of the **who** command (which you were introduced to in *Chapter 2*) and using it as input to the **wc** command (which counts lines, words, and/or characters) as follows:

```
who | wc -l <CR>
```

This example shows that the standard output of the **who** command was passed to the **wc -l** command (**-l** is the option that counts the number of lines output by the **who** command, each corresponding to a user who is logged into your UNIX system.)

Summary

Table 4-3 summarizes which operator performs which redirection task and what general format should be followed in using it.

TABLE 4-3
Options for Redirecting Input and/or Output†

Action	Operator	General Format
Redirecting output to a file	>	command > filename
Redirecting and appending output to a file	>>	command >> filename
Redirecting input from a file	<	command < filename
Redirecting output of first command to be input for second		command command

* See Chapter 7 of the *UNIX System Release 2.0 User's Guide*, available from AT&T for complete details on how to use these options.

† Blank spaces immediately before and after redirection operators are optional.

Running Multiple Programs

There are two methods for running multiple programs: you can specify more than one command to execute in sequence from a single command line or you can run commands simultaneously.

Executing Commands in Sequence

Up to this point, the command lines to which you have been introduced and examples for using them have dealt with asking the shell to run a single request or program. For example, each of the command lines `cat filename<CR>`, `date<CR>`, and `ls -l directoryname<CR>` requests the shell to perform one task. You can, however, ask the shell to execute more than one request per command line. Sequential execution allows you to enter as many commands as you wish on one command line and have them execute in the order in which you input them.

To do so, you should first be familiar with the general rules for command line syntax, given in the "Using the File System" chapter. Briefly, command line syntax orders elements in the command line so that the command name, any options you wish to specify, and the data on which the command is to operate (usually the name of a file or directory) follow one another.

To execute more than one command on a line, simply separate the request sequences with semicolons (;) as follows:

command option(s) argument(s); command option(s) argument(s); ...<CR>

For example, to determine where you are in the file system and then list the contents of the directory in which you are working, you can type `pwd; ls<CR>` and the terminal monitor might read:

```
$pwd; ls<CR>
/user1/starship/bin
dir
list
tools
$
```

As you can see, the output of the multiple commands is ordered the same way the input is: first, the current working directory is given (in response to `pwd`) and, then, the names of the files and/or directories it holds follow (in response to `ls`).

You could just as easily type `who am i; date; who<CR>` or `mkdir directoryabc; cd directoryabc; pwd<CR>` or any combination of commands that you wish to use.

Executing Commands Simultaneously

In addition to running programs sequentially, you can choose to run them simultaneously. To do so, you need to know the difference between foreground and background commands. When a program runs in the background, the computer is executing that program concurrently with the commands that you enter or with the program

that you run in the foreground. However, the computer considers your foreground work more important, in a sense, than your background program. This difference has no perceivable effect on the execution of most programs, but running a job in the background is a useful technique when you wish to run a lengthy or time-consuming job without tying up your terminal.

All the command lines used in this guide until now have been examples of foreground commands. This means that they were initiated and run to completion before other commands could be executed and before the shell would return the \$ prompt for you to continue. However, you also have the option of running a command in the background so you can continue to work in the foreground.

You can run a command in the background by placing an ampersand (&) at the end of the command line as follows:

command option(s) argument(s) &<CR>

When the shell reads the &, it starts running the program, prints an identification number, and displays the \$ prompt so you can use the terminal immediately for other work.

To save the output from the job you are running in the background, you must redirect the results of the execution into another file so you can look at or use the output when you are ready. For example, if you input the command **cat file1 file2 > file3 &<CR>**, the shell would first give you an identification number, and then the prompt. It will also save the results of **cat file1 file2** in a file named *file3*. When you are ready to peruse *file3*, simply use **cat** or **pg**. If you do not redirect the output, then no output is saved.

When a program is running in the background, it ignores interrupt and break signals, but if you log off, the shell terminates the background program along with the computing session. If you would like to stop a background command while you are still logged into the UNIX system, type **kill id<CR>**, where *id* is the identification number of the command. On the other hand, to have a program continue to run after you log off, you can use the **nohup** command (which stands for "no hang up") in the following way

nohup command &<CR>

When you do, the command will continue to run until completion and its output is saved in a file called *nohup.out* (which stands for *nohup* output).

Customizing Your Computing Environment

The information in this section deals with another dimension of control provided to you by the shell called your environment. When you log into the UNIX system, the shell automatically sets up a computing environment for you. You can choose to use it as supplied by the system or you can tailor it to meet your needs.

By default, the environment set up by the shell includes the variables:

HOME = your login directory,

PATH = route the shell takes to search for executable files or commands (typically *PATH=/bin:/usr/bin*), and

LOGNAME = your login name.

If you find the default environment satisfactory, simply leave it as it is and go on with your work. However, if you would like to modify it, you must have a file in your login directory named *.profile*. If you do not, you can create one with a text editor like *ed* or *vi*.

To determine if you have a *.profile*, move to your login directory and type *cat .profile<CR>* and its contents should appear on the terminal monitor. Typically, the *.profile* tests for mail and sets data parameters, system variables, and terminal settings.

Possible modifications to your login environment might include changing your login prompt, setting tab stops, and changing erase and kill characters. (If you would like to customize your *.profile*, see the section entitled CHANGING YOUR ENVIRONMENT in the "Screen Editor Tutorial" chapter.

COMMUNICATING ELECTRONICALLY

Before the days of office automation, you would probably have thought of relaying a message or information to someone either personally or by way of a letter, note, or telephone conversation. Now as a UNIX system user, you can choose to communicate electronically with other UNIX system users by way of the computer.

You can send messages or transmit information stored in files to other users who work on your system or on another UNIX system. To do so, your UNIX system must be able to communicate with the UNIX system to which you wish to send information. In addition, the command you use to send information depends on what you are sending.

This guide introduces you to these communication programs:

- mail* -- This command is typically used for sending messages to others and reading the messages sent to you. You can use **mail** to send messages or files to other UNIX system users using their login names as addresses. And, at your convenience, you can use the **mail** command to read messages sent to you by other users. With **mail**, the recipient can choose when to read it.

- uuto/uupick* -- These commands are used to send and retrieve files. You use the **uuto** command to send a file(s) to a public directory; when its available to the recipient, the person is sent mail telling him/her that the file(s) has arrived. The recipient then can use the **uupick** command to copy the file(s) from the public directory to the directory of choice.

- mailx* -- This command is a sophisticated, more powerful spin-off of **mail**. It offers a number of options for managing the electronic mail you send and receive.

The "Communications Tutorial" chapter teaches you how to use the **mail**, **uuto**, and **uupick** commands. It also introduces you to the **mailx** command so you can begin to use it.

PROGRAMMING IN THE SYSTEM

The UNIX system provides an efficient, effective, and convenient environment for programming and software development. This section briefly describes the environment and your programming options when working in it.

If you are not a programmer, your immediate reaction might be to skip this section. But you need not be a programmer or software developer to enjoy some of the capabilities that fall under the realm of programming.

For example, you can use the shell as a command level programming language as well as the command line interpreter. Shell programming capabilities are useful and usable techniques that allow you to take simple, existing programs and make them more powerful. So why not read on.

On the other hand, if you're interested in sophisticated programming and software development capabilities, this section can serve as a springboard to using them.

What you can expect to find in the next few pages is an overview of shell and C language programming and a mention of other languages that can be used on the UNIX system. In addition, an overview of some UNIX system tools for software development is included.

Programming in the Shell

Most interactive users of the UNIX system think of the shell solely as the command language interpreter. The shell, however, is also a command level programming language. What this means is that you can let the shell continue to act as your liaison with the computer or you can program the shell to repeat sequences of instructions and to test certain considerations for you automatically. When you program the shell to perform a task, you use the shell to read and to execute commands that you place in an executable file. These files are sometimes called shell scripts or shell procedures.

When you use the shell in this manner, it provides you with features, like variables, control structures, subroutines, and parameter passing

UNIX SYSTEM CAPABILITIES

that are very similar to those offered by programming languages. These features provide you with the ability to create your own tools by linking together system commands.

For example, you can write a simple shell procedure from existing UNIX system programs that tells you the date and time along with the number of users working on your UNIX system. One way to do so is illustrated in the following screen:

```
$ cat > users<CR>
date; who | wc -l<CR>
<^d>
$ chmod u+x users<CR>
$
```

A file called *users* is created using the `>` redirection operator. In the example, `cat` is taking as input everything you type after `<CR>` on the command line and placing it in a file named *users*. Then the file is made executable with the `chmod` command. If you type the command `users<CR>`, your terminal monitor would look something like the next screen.

```
$ users<CR>
Tues May 22 10:29:09 CDT 1984
 7
$
```

The output tells you that seven users were logged into the system when you typed the command at approximately 10:30 A.M. on Tuesday, May 22.

For additional information on shell procedures and for more sophisticated shell programming techniques, see other references on UNIX shell programming.

Programming in the C Language

C is a general purpose programming language. It is a relatively "low level" language, which means that C deals with the same sort of objects that most computers do, namely characters, numbers, and addresses. These may be combined and moved about with the usual arithmetic and logic operators.

C is closely associated with the UNIX system because it was developed on the UNIX system and because UNIX system software is largely written in C.

Although the C programming language is implemented on many computers, it is independent of any particular machine architecture. With a little care, it is easy to write portable programs, that is, programs that can be run without change on a variety of computers if the machine supports a C compiler.

The C programming language comprises the following main elements:

- *Types, operators, and expressions*--Constants and variables are the basic data objects manipulated in a program. Constants are data objects that do not change during the execution of a program, while variables are assigned new values throughout execution. Declarations list variables, state type, and perhaps initial values. Operators specify what is to be done on them. Expressions combine variables and constants to produce new values.
- *Control flow*--Control flow statements of a language specify the order in which computations are done. In C, these include *if-else*, *else-if*, and *switch* statements, and *while*, *for*, and *do-while* loops. In addition, *break*, *continue*, and *goto* statements can be used. Labels can be used as well.

UNIX SYSTEM CAPABILITIES

- *Functions and program structure*--C programs generally consist of numerous small functions rather than a few big ones. These functions break large computing tasks into smaller ones and enable you to build on what others have done.
- *Pointers and arrays*--A pointer is a variable that contains the address of another variable. Pointers are frequently used when programming in C because oftentimes they provide the only way to express a computation and partly because their use typically leads to more compact and efficient code than can be obtained in other ways.
- *Structures*--A structure is a collection of one or more variables, possibly of different types, that are grouped together under a single name for convenient handling. Structures help to organize complicated data because they permit a group of related variables to be treated as a unit instead of separate entities.
- *Input and output*--A standard I/O library containing a set of functions designed to provide a standard input and output system is available for C programs. This library is a UNIX system feature available for programming in C.

These elements are covered in detail in *The C Programming Language* by B.W. Kernighan and D.M. Ritchie (Prentice-Hall, 1978). Additional information is also available in the Software Development System manual.

Other Programming Languages

In addition to C, other programming languages are available for use on the UNIX system, such as FORTRAN-77, BASIC, Pascal, COBOL, APL, LISP, and SNOBOL.

You can obtain details on FORTRAN-77 in the Software Development System manual. Or contact Microport Systems for document availability and ordering information on the others.

Tools to Aid Software Development

This section highlights some sophisticated software development tools available on the UNIX system. The tools are designed to make development of software easier and to provide you with a systematic approach to programming.

There are numerous software development aids provided by the UNIX operating system. This section introduces you to five of them to give you an idea of what you can expect development utilities to do. They are:

SCCS -- Source Code Control System,

RJE -- Remote job entry,

make -- Maintaining programs,

lex -- Generating programs for simple lexical tasks, and

yacc -- Generating parser programs.

Refer to the Software Development System manual for more information.

Source Code Control System (SCCS)

The Source Code Control System (SCCS) is a collection of UNIX system commands that helps you to control and report changes to source code files or text files. SCCS allows you to access different versions of the same file while maintaining only one file. The way this works is that SCCS stores the original file on a disk. Whenever modifications are made to the file SCCS stores only those changes as a set in something called a *delta*. Each delta or set of changes is numbered to reflect the different versions of a file. You can then choose to retrieve either the original file or a version of the original file.

UNIX SYSTEM CAPABILITIES

By allowing SCCS to store and control all iterations of a file, space allocations for storage are minimized and administration of different versions of the same program or document is efficient and simplified. Updates to files can be made quickly and the original version of a program or document is retained if you should need to recall it later.

For additional information,, see the "SCCS User's Guide" in the Software Development System manual.

Remote Job Entry (RJE)

Remote job entry (RJE) is a software package designed to facilitate communication between a UNIX operating system and an IBM System/360 or an IBM System/370 computer. The RJE software allows the UNIX operating system to communicate with the IBM Job Entry Subsystem by mimicking an IBM System/360 remote multileaving work station. A set of background processes support RJE, and the UNIX system uses these processes to submit jobs for remote execution on the networked IBM system.

When RJE software runs, it does so in the background. It transmits jobs (consisting of job control statements [JCL] and input data) that you queue with the `send` command and status reports you request with the `rjstat` command. In turn, the RJE software subsystem receives print and punch data sets and message output from the IBM system.

Maintaining Programs (make)

The `make` command is a tool for maintaining, supporting, and regenerating large programs or documents on the basis of smaller ones. Since it is easier to handle and modify small programs, it is recommended that if you wish to develop a large program, you start by creating a series of smaller ones that work together to produce the large one.

The **make** command provides you with a method to store all the information you need to assemble small programs or modules into a large, more sophisticated one. A file called a **makefile** holds the file names of the small programs, the steps necessary to generate the large program, and specifies the dependencies among the files.

When **make** executes the **makefile**, the date and time you last modified any of the small programs are checked and the operations needed to update them are performed in sequence. Then, **make** goes on to create the overall large program.

For details on the operation of **make**, see the "Make" and "Augmented Version of Make" chapters in the Software Development System manual.

Generating Programs for Lexical Tasks (*lex*)

The **lex** utility generates programs to be used in simple lexical analysis of text. Lexical analysis is done by evaluating a stream of characters and constructing the forms that are acceptable to the language. Proper forms are defined in the **lex** program and usable forms can be defined by **lex** defaults or by you. **Lex** produces a subroutine as output that must be compiled and combined with other programs to use the lexical analyzer.

The processing done by the **lex** command can be the first step in creating a compiler-type program. In addition, it can be useful as a preprocessing tool for many different software generation functions.

For additional information on the **lex** command, see the "Lexical Analyzer Generator" in the Software Development System manual.

Generating Parser Programs (*yacc*)

The **yacc** program, short for yet another compiler compiler, is primarily used in the generation of software compilers. Essentially, **yacc** is a utility for creating parser subroutines. The way this works is that first **yacc** uses specified syntax and produces source code for a parser subroutine. Then, the parser subroutine is compiled, and finally used with a program that calls it to parse input. In this way,

UNIX SYSTEM CAPABILITIES

structure can be imposed on the input to a program and the desired language can be created from defined rules.

See the "Yacc" chapter in the Software Development System manual for details on the yacc command. Or refer to section "1" of the Runtime System manual.

Chapter 5

SINGLE USER AND MULTIUSER

INTRODUCTION

There are two main modes of operation of a UNIX operating system: single user (level S) and multiuser (level 2). The run level has 8 possible values: 0-6, and s (or S). Single user is always s or S. Although multiuser is normally level 2, your system administrator can configure the `/etc/inittab` file to run multiuser at any level from 0 to 6. Furthermore, the `inittab` file can be configured so that certain procedures are followed automatically only the first time a certain run level is entered. For example, normally you will be asked to verify date and file systems the first time you change your system to multiuser. This is caused by an entry in the `inittab` file. Subsequent changes in run level will not perform this procedure automatically unless you specifically change the `inittab` file. For more information on `init` refer to `init(1M)` in the Runtime System manual, `inittab(4)` in the Software Development System manual, or consult your local UNIX system administrator.

When in single user mode, all dial-up ports and hard-wired terminals are disabled and only the console terminal may interact with the processor. This mode of operation allows you to make necessary changes to the system without any other processing taking place. However, you will normally run the UNIX operating system in multiuser mode. Consult the chapter for your processor before proceeding with any of these procedures.

SINGLE USER ENVIRONMENT

In single user mode, you may type any available UNIX system command (followed by a `<cr>`). When the system has completed execution of the command, it will prompt with the `"#"` again on the next line. You use the single user environment primarily to do filesaves, system maintenance, modification, or repair operations.

The typical sequence of commands to change the system to multiuser mode is:

- **fsck**
- **telinit 2**

The fsck Command

The command **fsck** will interactively repair any damaged file systems that result from a crash of the operating system. You should also use it to ensure that the file systems are not damaged before going into multiuser mode or taking filesaves. Usually, you will want to respond "yes" to all the prompts; however, in the event of a system crash, the damage may be extensive enough to warrant recovery from a backup pack. The procedure for this is discussed under *FILESAVES* in this chapter. See **fsck** in the Runtime System manual for details on the various options available and the different errors that can occur.

An example of a check of a consistent file system is illustrated below:

```
# fsck /dev/rdisk/0s6
/dev/rdisk/6s1
File System: usr Volume: p0603
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
2441 files 16547 blocks 31889 free
#
```

A file system that has been damaged can be repaired as shown below. The **y** is your response. When checking a file system, you can avoid the questions asked by **fsck** concerning inconsistencies found by using the **y** option. This option will automatically attempt repairs as though you answered **yes** to the questions. Use this with caution—the corrections usually involve some data loss. Use the following example if you decide to interactively repair the file system.

```
# fsck /dev/rdisk/6s0
```

The UNIX operating system responds:

```
/dev/rdisk/0s6
File System: fs1 Volume: p0603
** Phase 1 - Check Blocks and Sizes
POSSIBLE FILE SIZE ERROR I=2500
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
UNREF FILE I=2500 OWNER=255 MODE=100755
SIZE=0 MTIME=Dec 31 1983 1983
CLEAR? y
** Phase 5 - Check Free List
2441 files 16547 blocks 889 free
***** FILE SYSTEM WAS MODIFIED *****
#
```

All mountable file systems should be listed in the file `/etc/checklist` which `fsck` uses, and you should check these file systems each time the system is rebooted.

A faster alternative to using `fsck` is `checkall`. The `checkall` command uses `dfsk` (a front end for `fsck`) to simultaneously check two file systems in different disk drives. Therefore, you can check file systems faster with `checkall` than you can with `fsck`. Included in `checkall` are the file system names that normally appear in `/etc/checklist` (see `checkall` in Section "1" of the Runtime System manual).

WARNING: Never execute `fsck` on a mounted file system; it will have a bad effect since you are repairing only the physical disk. The only exception to this is the root file system, which is always mounted.

An example of repairing the root file system follows:

```
# fsck /dev/dsk/0s6
/dev/dsk/0s6
File System: root Volume: p0001
** Phase 1 - Check Blocks and Sizes
POSSIBLE FILE SIZE ERROR I=416
```

SINGLE USER AND MULTIUSER

```
POSSIBLE FILE SIZE ERROR I=610
POSSIBLE FILE SIZE ERROR I=614
POSSIBLE FILE SIZE ERROR I=618
POSSIBLE FILE SIZE ERROR I=625
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
UNREF FILE I=416 OWNER=uucp MODE=100400
SIZE=0 MTIME=Nov 20 16:23 1983
CLEAR? y
UNREF FILE I=610 OWNER=csw MODE=100400
SIZE=0 MTIME=Nov 20 16:26 1983
CLEAR? y
UNREF FILE I=625 OWNER=cath MODE=100400
SIZE=0 MTIME=Nov 20 16:26 1983
CLEAR? y
FREE INODE COUNT WRONG IN SUPERBLK
FIX? y
** Phase 5 - Check Free List
1 DUP BLKS IN FREE LIST
BAD FREE LIST
SALVAGE? y
** Phase 6 - Salvage Free List
585 files 5463 blocks 4223 free
***** BOOT UNIX (NO SYNC !) *****
#
```

At this time you must immediately halt the processor and then reboot the system (see the chapter concerned with operations on your processor for start-up procedures).

The telinit 2 Command

After you have checked the file systems, you may change the UNIX operating system to multiuser. Do this by entering the command **telinit 2**. This command activates processes that allow users to log in to the system, turn on the accounting and error logging, mount any indicated file systems, and start the **crond** and any indicated daemons. Depending upon the type of data set your site has, you may have to manually flip the toggles or pop the buttons on the data sets to allow users to log in.

MULTIUSER ENVIRONMENT

There are two ways to get to this level: by typing `telinit 2`; or, specifying a run level of 2 after the boot. Users are permitted to access all mounted file systems and execute all available commands. In this mode, you can perform file restore procedures and take periodic status checks of the system. Some of these periodic status checks can include:

- A check of free blocks (**df**) remaining on all mounted file systems to ensure that a file system does not run out of space.
- A check on **mail** to root or whatever login receives requests for file restores.
- A check on the number of users on the system (**who**).
- A check of all running processes (**ps -eaf** or **whodo**) to determine if there is some process using an abnormally large amount of CPU time.

If your site has other run levels defined, you can use the `telinit` command to change to those run levels. Finally, to change a multiuser system to single user, refer to *SYSTEM SHUTDOWN* in this chapter, and to Shutdown (1M).

```

fs:::sysinit:/etc/bcheckrc </dev/console >/dev/console 2>&1
mt:::sysinit:/etc/brc </dev/console >/dev/console 2>&1
ck:::sysinit:/etc/bsetdate </dev/console >/dev/console 2>&1
is.2: initdefault: # iAPX286 @ (#) inittab.sh 1.3
pf:::powerfail:/etc/powerfail >/dev/console 2>&1 #power fail routines
s0:056: wait:/etc/rc0 </dev/console >/dev/console 2>&1
s2:2: wait:/etc/rc2 </dev/console >/dev/console 2>&1
s3:3: wait:/etc/rc3 </dev/console >/dev/console 2>&1
d1:056: wait:/bin/kill -15 -1 > /dev/console </dev/console 2>&1
d2:056: wait: sleep 5
d3:056: wait:/bin/kill -9 -1 > /dev/console 2>&1
d4:056: wait: sleep 5
d5:056: wait:/etc/fumountall > /dev/console 2>&1
d6:056: wait: echo '\nThe system is down.' > /dev/console
r0:0: wait:/etc/uadmin 2 1 # halt and reboot if possible
r5:5: wait:/etc/uadmin 2 2 # return to firmware if possible
r6:6: wait:/etc/uadmin 2 1 # reboot if possible
co:1234: respawn:/etc/getty console console
c1:2: respawn:/etc/getty cons1 virtcon
c2:2: respawn:/etc/getty cons2 virtcon
c3:2: respawn:/etc/getty cons3 virtcon
t0:2: off:/etc/getty tty0 9600
t1:2: off:/etc/getty tty1 9600
t2:2: off:/etc/getty tty2 9600
t3:2: off:/etc/getty tty3 9600
t4:2: off:/etc/getty tty4 9600
t5:2: off:/etc/getty tty5 9600
t6:2: off:/etc/getty tty6 9600
t7:2: off:/etc/getty tty7 9600

```

Figure 5-1. Sample inittab with no serial ports enabled.

from copying this file, you could deny them read permission by typing `chmod go-r display<CR>`. The *g* and *o* stand for group members and all other system users, respectively, and the *-r* denies them permission to read or copy the file. Check the results with the `ls -l` command.

```
$ chmod go-r display<CR>
$ ls -l<CR>
total 35
-rwx--x--x 1 starship project 9346 Nov 1 08:06 display
-rwx--x--x 1 starship project 6428 Dec 2 10:24 list
drwx--x--x 2 starship project 32 Nov 8 15:32 tools
$
```

A Note on Permissions and Directories. If you read the preceding pages describing the `chmod` command, you might have gathered that you can use this command to grant or deny permission for directories as well as files. It is true, you can. To do so, simply use the directory name instead of a file name on the command line.

The impact, however, of granting or denying permissions for directories to various system users is worth considering. For example, if you grant read permission for a directory to yourself (*u*), members of your group (*g*), and other system users (*o*), every user who has access to the system can read the names of the files that directory contains by using the `ls -l` command. Similarly, granting write permission allows the designated users to create new files in the directory and change and remove existing ones. And granting permission to execute the directory allows the designated users the ability to move to that directory (and make it their working directory) by using the `cd` command.

An Alternate Method. The `chmod` method described in the preceding pages is one of two ways to change permissions to read, write, and execute files and directories. The method previously described uses symbols, such as *r*, *w*, *x* and *u*, *g*, *o*, to specify instructions to `chmod`. Hence, it is called the symbolic method.

The alternate method uses a number system called octal that is different than the decimal number system we typically use on a day-to-day basis. This method uses three octal numbers ranging from 0 through 7 to assign permissions. If you wish to use the octal method when changing permission, see the description of **chmod** in Section 1 of this manual.

Summary. The command recap that follows provides a quick reference on how **chmod** works.

Command Recap

chmod - change permission modes for files (and directories)

<i>command</i>	<i>instruction</i>	<i>arguments</i>
chmod	who + - permission	filename(s) directoryname(s)
Description:	chmod gives (+) or removes (-) <i>read</i> , <i>write</i> , and <i>execute</i> permissions for three types of system users: <i>user</i> (you), <i>group</i> (members of your group), and <i>other</i> (all other users able to access the system on which you are working).	
Remarks:	The instruction set can be represented in either octal or symbolic terms.	

Advanced Commands

You will become more and more familiar with the file system as you use the commands thus far discussed in this chapter. As this familiarity increases so might your need or interest for more

sophisticated information processing techniques when working with files. This section introduces you to three commands that give you just that. These commands and their capabilities are listed as follows:

diff -- Finds difference between two files,

grep -- Searches a file for a pattern, and

sort -- Sorts and merges files.

The following discussion only scratches the surface on information processing techniques available with the UNIX system. You may refer to section 1 of the Runtime System manual for additional information.

C-6

Identifying Differences Between Files (diff)

The **diff** command locates all the differences between two files and proceeds to tell you how to change the first file to be a carbon copy of the second. It reports all differences between the files.

The basic format for the command is:

```
diff file1 file2<CR>
```

If *file1* and *file2* are identical, the system returns the \$ prompt to you. If not, the **diff** command instructs you on how to bring the first file into agreement with the second by using line editor (**ed**) commands. The UNIX system flags lines in *file 1* with the < symbol and the *file 2* with the > symbol.

USING THE FILE SYSTEM

For example, if you use the **diff** command to identify differences between the files *johnson* and *sanders*, the system would respond as follows:

```
$ diff johnson sanders<CR>
2,3c2,4
< to Mr. Johnson on the topic of
< office automation.
--
> to Mrs. Sanders inviting her to
> speak at your departmental
> meeting.
$
```

The first line of the system response is

2,3c2,4

which means lines 2 through 3 in the file *johnson* must be changed (designated by *c*) to lines 2 through 4 in the file *sanders*. The system then displays lines 2 through 3 in the file *johnson* as follows:

```
< to Mr. Johnson on the topic of
< office automation.
```

and lines 2 through 4 in the file *sanders*

```
> to Mrs. Sanders inviting her to
> speak at our departmental
> meeting.
```

If you make these changes (using the **ed** or the **vi** text editing program), the file *johnson* will be identical to the file *sanders*. Remember, the **diff** command tells you exactly what the differences are between the named files. If you simply want an identical copy of a file, use the **cp** command.

Refer to the recap that follows for a summary of what you can expect the **diff** command to do when no options are specified. See the reference to section 1 of the Runtime System manual for details on available options.

Command Recap

diff - finds differences between two files

<i>command</i>	<i>options</i>	<i>arguments</i>
diff	available*	file1 file2

Description: **diff** reports what lines are different in two files and what you must do to make the first file identical with the second.

Remarks: Instructions on how to change a file to bring it into agreement with another file are line editor (ed) commands: *a* (append), *c* (change), or *d* (delete). Numbers given with *a*, *c*, or *d* indicate the lines to be modified. Also used are the symbols *<* (indicating a line from the first file) and *>* (indicating a line from the second file).

* See section 1 of the Runtime System manual for all available options and an explanation of their capabilities.

Searching a File for a Pattern (*grep*)

You can request the UNIX system to search through files for a specific word, phrase, or group of characters by using the **grep** command. Technically, **grep** means globally search through a file or files to locate a regular expression and print the lines that contain the regular expression. Put simply, a regular expression is the pattern of characters--be it a word, a phrase, or an equation--that you stipulate.

The basic format for the command line is:

grep pattern file(s) <CR>

USING THE FILE SYSTEM

Thus, to locate the line containing the word *automation* in the file *johnson*, you would type:

```
grep automation johnson<CR>
```

and the system would respond as follows:

```
$ grep automation johnson <CR>
office automation
$
```

The output gives you all the lines in the file *johnson* that contain the pattern for which you were searching, which is the word *automation*.

If the pattern contains multiple words or any characters that have a special meaning to the UNIX system, such as \$, |, *, ?, and so on, the entire pattern must be enclosed in single quotes. (For a complete explanation of the special meaning for these and other characters see an outside reference work on shell programming.) For example, if you are interested in locating the lines containing the pattern *office automation*, the command line and system response would read:

```
$ grep `office automation` johnson<CR>
office automation.
$
```

But what if you could not recall to whom you sent a letter on the topic of office automation in the first place--Mr. Johnson or Mrs. Sanders? You could type:

```
grep `office automation` johnson sanders<CR>
```

If you did, the system would respond in the following manner:

```
$ grep `office automation` johnson sanders<CR>
johnson:office automation.
$
```

The output tells you that the pattern *office automation* is found once in the file *johnson*.

In addition to the capabilities of the `grep` command that are summarized in the recap that follows, the UNIX system provides variations to the basic `grep` command, called `egrep` and `fgrep`, along with several options that further enhance the searching powers of the command. See section 1 of the Runtime System manual if you are interested in learning more.

C-6

Command Recap

grep - searches a file for a pattern

<i>command</i>	<i>options</i>	<i>arguments</i>
grep	available*	pattern file(s)

Description: `grep` searches the file or files you name for lines containing a pattern and then prints the lines that match. If you name more than one file, the name of the file containing the pattern is given also.

Remarks: If the pattern you give contains multiple words or special characters, enclose the pattern in single quotes on the command line.

* See section 1 of the Runtime System manual for all available options and an explanation of their capabilities.

Sorting and Merging Files (*sort*)

The UNIX system provides you with an efficient tool called **sort** for sorting and merging files. The basic form of the command line is:

sort file(s)<CR>

which causes lines in the specified files to be sorted and merged in the order defined by the ASCII representations of the characters in the lines.

- Lines beginning with numbers are sorted by digit and listed before letters in the output,
- Lines beginning with uppercase letters are listed before lines beginning with lowercase letters, and
- Lines beginning with symbols, such as *****, **%**, or **@**, are sorted on the basis of the symbol's ASCII representation.

To get an idea of how the **sort** command works, let's say that you have two files, named *phase1* and *phase2*, each containing a list of names that you wish to sort alphabetically and finally interfile into one list. First, display the contents of each file using the **cat** command.

```
$ cat phase1<CR>
Smith, Allyn
Jones, Barbara
Cook, Karen
Moore, Peter
Wolf, Robert
$ cat phase2<CR>
Frank, M. Jay
Nelson, James
West, Donna
Hill, Charles
Morgan, Kristine
$
```


(Note: we could have used the command line `cat phase1 phase2<CR>` instead of listing the contents of each file separately.)

Now, sort and merge the contents of the two files using the `sort` command. Note that the output of the `sort` program will print on the terminal monitor unless you specify otherwise.

```
$ sort phase1 phase2<CR>
Cook, Karen
Frank, M. Jay
Hill, Charles
Jones, Barbara
Moore, Peter
Morgan, Kristine
Nelson, James
Smith, Allyn
West, Donna
Wolf, Robert
$
```

In addition to putting together simple listings as in the previous examples, the `sort` command can rearrange the lines and parts of lines (called fields) according to a number of other specifications you can designate on the command line. The possible specifications are complex and are not within the scope of this text. You should consult section 1 of the Runtime System manual for a full rundown on the available options.

However, the following command recap summarizes the capabilities of the `sort` program.

Command Recap

sort - sorts and merges files

<i>command</i>	<i>options</i>	<i>arguments</i>
sort	available*	file(s)

Description: **sort** sorts and merges lines from the file or files you name and displays the result on your terminal.

Remarks: If no options are specified on the command line, lines are sorted and merged in the order defined by the ASCII representations of the characters in the lines.

* See section 1 of the Runtime System manual for all available options and an explanation of their capabilities.

SUMMARY

This chapter described the structure of the file system and presented ways to use and to navigate through the file system via UNIX system commands. The "UNIX System Capabilities" chapter gives you an overview of a variety of UNIX system capabilities; such as text editing, using the shell as a command language, communicating electronically with other system users, and programming and developing software.

Chapter 7

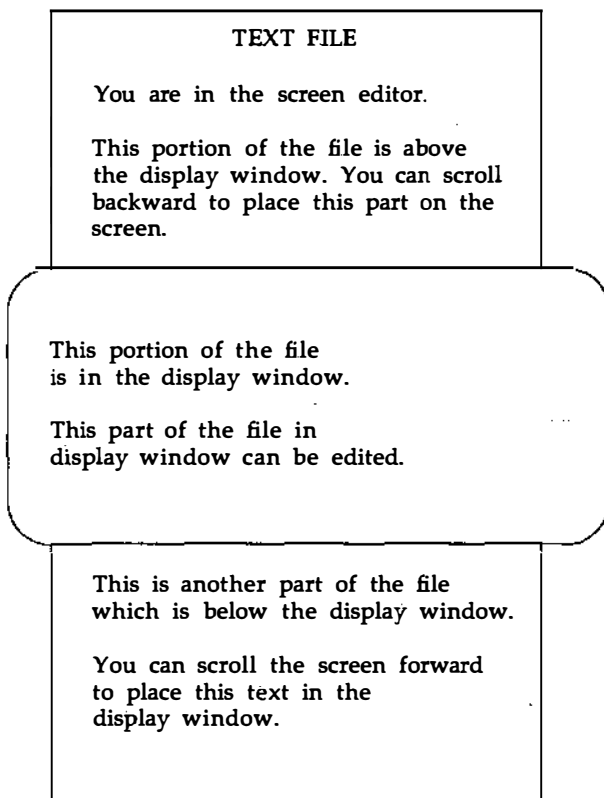
SCREEN EDITOR TUTORIAL (vi)

GETTING ACQUAINTED WITH vi

The screen editor, accessed by the **vi** command, is a powerful and sophisticated tool for creating and editing files. The video display terminal is used as a window to view the text of a file. Within this window, you can add, delete, or change text in much the same way as you would on a typewriter or with paper and pencil. However, making corrections in **vi** does not involve white out, correction tape, or cutting and pasting. A few simple commands change the text, and these changes are quickly reflected in the text on the screen.

The **vi** editor displays from 1 to several lines of text. The cursor can be moved to any point on the screen and text can be created, changed, or deleted from that point. The text in the file can be scrolled forward to reveal the lines below the current window, the window that is on the screen now. Or, the file can be scrolled backward to reveal lines above the current window. (See the display on *page 6-2*.) Other commands can place you at the beginning or end of the file, paragraph, line, or word.

Besides the convenience of editing portions of text within the window, **vi** also gives you the advantage of some line editor commands, such as the powerful global commands that make the same change throughout the whole file.



Editing window of vi displaying part of a file

HOW TO READ THIS TUTORIAL

This chapter is a tutorial on how to access and use **vi**. Although there are more than 100 commands within **vi**, this tutorial covers only the basic commands that will enable you to effectively use **vi**. The following basics will be covered:

- How to set up your particular type of terminal so you can access **vi**.

- How to get started creating a file, deleting some of your mistakes, writing the text into a UNIX system file, and then leaving **vi** to go back to the shell command mode,
- How to move around within the file, so that you can create, delete, or change text,
- How to electronically cut and paste your text,
- How to use some special commands and shortcuts,
- How to temporarily escape to the shell to perform some shell commands and then return to edit the current window of text,
- How to use some line editing commands within **vi**,
- How to quit **vi**,
- How to edit several files in the same session,
- How to recover a file lost by an interruption to an editing session, and
- How to change your shell environment to automatically set your terminal configuration, and set an automatic carriage return.

In this tutorial, commands printed in **bold** should be typed into the system exactly as shown. UNIX system responses to those commands are printed in *italic*. The **vi** editor commands that do not print out on the screen will be enclosed in **<>**. For example, **<CR>** denotes carriage return, meaning press the RETURN key.

The **vi** editor has several commands executed by holding down the "control" or CTRL key while you press another key. These are called control characters. A **^** and a letter denote a control character in the text. For example, **^d** means hold down the control key and press the "d" key. Since **^d** is a command that does not appear on the screen, it will appear in the text as **<^d>**, meaning you should execute **vi** command **<^d>**. As you read the text you may want to glance back for a quick review of these conventions, which are summarized next.

bold command	(Type in exactly as shown.)
<i>italic response</i>	(The system's response to a command.)
roman	(Text that is being typed in a file.)
<CR>	(Commands that are typed in, but not reflected on the screen are enclosed in < >.)
^g	(A control character. Hold down the control key, CTRL, while you press "g".)

In the following sections, a full or partial screen may be used to display the examples showing how the commands are executed. An arrow will point to the letter that is over the cursor. Cursor movements on the screen are depicted by arrows pointing in the direction that the cursor will move.

The keys on your keyboard may be depicted as shown in the example of the "m" key.



Notice that the letter on the key appears as it does on your keyboard. However, when you press the key it will appear in lowercase in your text. If you need an uppercase letter, the example will include the SHIFT key.

The commands discussed in each section are reviewed at the end of the section. A summary of all the vi commands is found in vi (1).

At the end of some sections, exercises are given for you to experiment with those commands covered in the section. The answers to all of the exercises are at the end of this chapter.

GETTING STARTED

The best way to learn **vi** is to log into the UNIX system and do the examples and the exercises as you read the tutorial. If you experiment with the commands, they will become familiar to you and you will soon be adept at editing in **vi**.

You should be logged into the UNIX system, and ready to create a file in your current directory, the directory you are in now.

How to Set Terminal Configuration

Before you access **vi**, you must set your terminal configuration. That is, you must tell the system what kind of terminal will display the editing window of your file. Each type of terminal has a code name that can be recognized by the system. The code for your terminal is in the UNIX system file */etc/termcap*. The *termcap* file contains information about different terminals. You only need to know the code for your terminal, which is the first two letters of the line containing information about your terminal.

To find the code for your type of terminal, use the **grep** command to search the */etc/termcap* file for your terminal type. For example, if you have a TELETYPE 5420 terminal, type in the following from your login directory:

```
$ grep "teletype 5420" /etc/termcap<CR>
T7|5420|tty5420|teletype 5420 80 columns:
$
```

The code for a Teletype 5420 is T7.

To set the terminal configuration, type in:

```
TERM=code<CR>  
export TERM<CR>
```

TERM must be typed in uppercase and there are no spaces on either side of the equal sign. "code" will be the first two letters on the line for your terminal from the *termcap* file. In this command sequence, the **export** command assigns the terminal type to your login environment for this session while you are logged in to the UNIX System. You can learn more about exporting variables such as **TERM** in an outside reference on Bourne shell programming.

In the example below, you have logged into the UNIX system and have gotten your \$ prompt from the system. Then, you set your terminal configuration for the Teletype 5420.

```
$ TERM=T7<CR>  
$ export TERM<CR>  
$
```

Look up your terminal code in the *termcap* file, or ask your system administrator for the code. If you set your terminal configuration now, you can do the examples as you read the text.

Do not experiment typing in terminal configurations that do not match your terminal, since you may confuse the UNIX system, and you will either have to log off, hang up, or get the help of the system administrator to restore your login environment.

Later in this chapter, you will learn how to set your shell environment so that you do not have to set the terminal configuration each time that you log in to the UNIX system.

How to Access vi

Now you are ready to access vi.

Type in: **vi filename<CR>**

where *filename* is the name of the file you wish to edit, or the name of the file you are about to create.

After you have set your terminal configuration, you want to create a file called *stuff*. For the purpose of this example, **TERM** is set to T7.

```
$ TERM=T7<CR>
$ export TERM<CR>
$ vi stuff<CR>
```

The **vi** command will clear the screen and display the window for the screen editor. It should look like this:

```

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~

"stuff" [new file]
```

The vi editor window initially displays some lines of text. In this example there are no lines of text. The screen editor displays a ~ on each line to indicate the file is empty. The cursor is at the beginning

of the file waiting for the first command. In this example, the cursor appears as a short line. Your video display terminal may indicate the cursor by a blinking line or a reverse color block.

Problem:

If you access **vi** and get the following message you have forgotten to set the terminal configuration.

```
$ vi stuff<CR>
I don't know what kind of terminal you are on - all I have is unknown
[Using open mode]
"stuff" [New file]
```

Type in: **:q<CR>**

This returns you to the shell command mode, now you can set your terminal configuration.

How to Create Text

If you have successfully accessed **vi**, you are in the command mode of the screen editor, and **vi** is waiting for your commands. How do you create some text?

- Press the "a" key, **<a>**. Now you are in the append mode of **vi**. You can add text to the file. The **a** does not print out on the screen.
- Start typing in some text.
- To begin a new line press the carriage return key **<CR>**.
- Notice as you get close to the right margin a bell sounds to remind you to press the carriage return. Terminals which do not have a bell, may warn you another way, such as flashing the screen.

It is possible to set the carriage return so that it is automatic; this is discussed later in this chapter in the section on changing your environment.

How to Leave the Append Mode

If you are finished creating text, you need to leave the append mode and return to the command mode of **vi** to edit any text you have created, or to write the text into a UNIX system file. Press the escape key, **ESC** or **DEL**, denoted by **<ESC>**. You are now back in the command mode.

```
<a>  
Create some text <CR>  
in the screen editor <CR>  
and return to the <CR>  
command mode. <ESC>  
-  
-  
-  
-  
-  
-
```

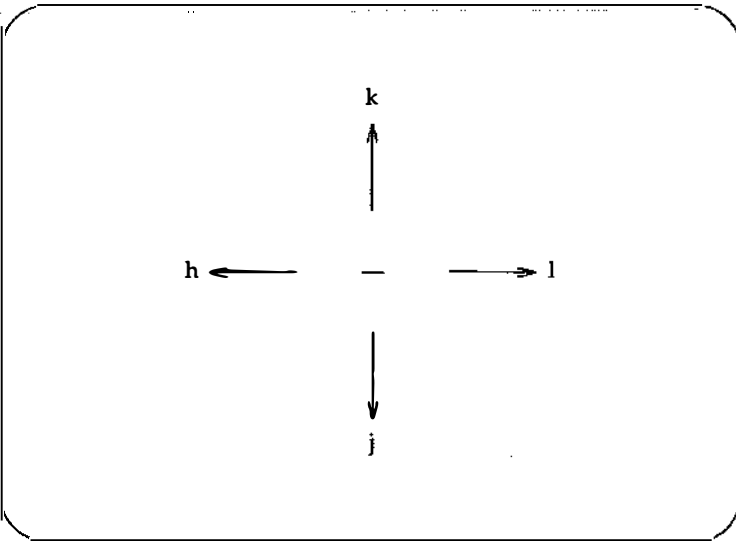
Problem:

If you press **<ESC>** and a bell sounds, **vi** is telling you that you are already in command mode. It will not affect the text in the file if you press **<ESC>** several times. The **vi** editor will only sound a bell each time that you press **<ESC>**.

How to Move the Cursor

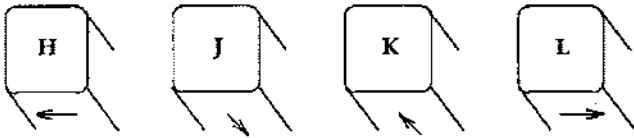
To edit your text, you need to move the cursor to the point on the screen where you will begin the correction. This is easily done with four keys that are next to each other on the keyboard, "h, j, k, l".

- <h> Moves the cursor one character to the left.
- <j> Moves the cursor down one line.
- <k> Moves the cursor up one line.
- <l> Moves the cursor to the right one character.



Right now try moving the cursor around. Watch the cursor on the screen while you press the keys <h>, <j>, <k>, and <l>. If you want to move two spaces to the right, press <l> twice. If you want to move up four lines, press <k> four times. If you cannot go any farther in the direction you have indicated, vi will sound a bell.

Many people who use vi find it helpful to mark these four keys with arrows indicating the direction that each key moves the cursor. Mark an arrow on each of four small pieces of white correction tape and place a left arrow on the front of the "h" key, a down arrow on the "j" key, an up arrow on the "k" key, and a right arrow on the "l" key.



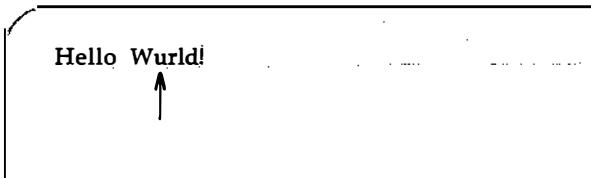
Some terminals have special cursor control keys that are marked with arrows. These may be used as "h, j, k, and l" keys are used.

Problem:

If you are trying to move the cursor around on the screen and the letters h, j, k, and l print out on the screen, you are still in the append mode of vi. Press <ESC>. Most of the commands in the screen editor are silent, that is they do not print out. If the screen editor commands are printing out on the screen you are still in append mode. Press <ESC> and try the commands again.

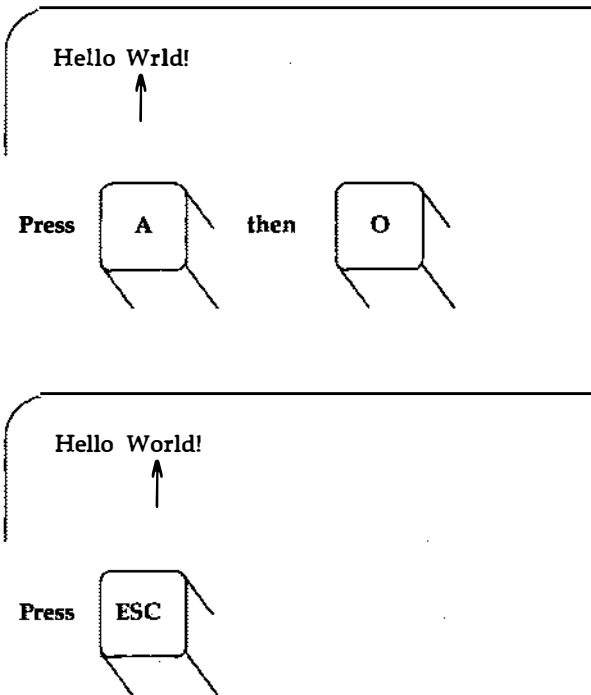
How to Delete Text

If you have put in an extra character in the text, you will want to delete that character. Move the cursor to that character, and press the "x" key. Watch the screen. The letter will disappear and the line will readjust to the change. If you want to erase three letters in a row, press <x> three times. In the examples below, the position of cursor is depicted by the arrow under the letter.



How to Add Text

If you need to add text at a certain point in the text that is in the window, move the cursor to that point using <h>, <j>, <k>, and <l>. Then, press <a> and text will be created after that point. As you append text, the characters to the right will move over on the screen to make room for the new characters. The vi editor will continue adding all characters that you type in, until you press <ESC>. If necessary the characters to the right will even wrap around onto the next line.



Moving around on the screen, or scrolling through the file to add or delete characters, words, or lines, is discussed in detail later in this tutorial.

How to Quit vi

The `vi` command creates a temporary buffer for you. This is equivalent to giving you a piece of scratch paper. When the text or data on the scratch pad is in the form you want for this editing session, you must write it to a UNIX system file. If you are done editing your test file, you will want to put this file in a file called *stuff* in the current directory and get back into the shell command mode.

Hold down the SHIFT key and press the "z" key twice, `<ZZ>`. The `vi` editor remembers the file name given to the `vi` command at the beginning of the editing session, and moves the text from the buffer of the editor to the file named *stuff*. You will get a notice at the bottom of the screen giving the file name, and the number of lines and characters in the file. Then, you are returned to the shell command level, and the UNIX system displays the shell prompt `$`. Since *stuff* is a new file, the notice at the bottom of the screen will include this fact.

```

<a>
This is a test file. <CR>
I am adding text to <CR>
a temporary buffer and <CR>
now it is perfect. <CR>
I want to write this file, <CR>
and return to the shell command <CR>
mode. <ESC><ZZ>
-
-
-
-
"stuff" [New file] 6 lines, 151 characters

$

```

SUMMARY OF GETTING STARTED

TERM=code	
export=TERM	Set the terminal configuration.
vi filename	Enter vi editor to edit the file called <i>filename</i> .
<a>	Add text after the cursor.
<h>	Move one character to the left.
<j>	Move down one line.
<k>	Move up one line.
<l>	Move to the right one character.
<x>	Delete a character.
<CR>	Carriage return.
<ESC>	Leave the append mode, and return to vi command mode.
<ZZ>	Write to a file, and quit vi.
:q	Quit vi.

EXERCISE 1

There is often more than one way to perform a task in vi. If the way you tried worked, then your answer is correct. Watch the screen as you give the commands, and see how it changes or how the cursor moves.

The answers to the exercises are at the end of this chapter.

- 1-1. If you have not logged in yet, do so now, and set your terminal configuration.

- 1-2. Enter vi and append the following five lines of text to a new file called *exer1*.

```
This is an exercise!  
Up, down  
left, right,  
build your terminal's  
muscles bit by bit.
```

- 1-3. Move the cursor to the first line of the file and the seventh character from the right. Notice as you move up the file, the cursor moves "in" to the last letter of the file, but it does not move "out" to the last letter of the next line.
- 1-4. Delete the seventh and eighth character from the right.
- 1-5. Move the cursor to the last line of the text, and the last character of that line.
- 1-6. Append a new line of text.

```
and byte by byte
```

- 1-7. Write the buffer to a file and quit vi.
- 1-8. Reenter vi and append two more lines of text to the file *exer1*.

What does the notice at the bottom of the screen say once you have reentered vi to edit *exer1*?

POSITIONING THE CURSOR IN THE WINDOW

Until now you have been positioning the cursor with the keys "h, j, k and, l". However, there are several commands to help you move the cursor quickly around the window.

This section on positioning the cursor in the window will look at:

- Positioning by characters on a line,

- Positioning by lines,
- Positioning by text objects
 - By words,
 - By sentences, and
 - By paragraphs, and
- Positioning in the window.

There are also several commands that position the cursor within the **vi** editing buffer. These commands will be looked at in the next section, *Positioning in the File*.

The **vi** editor provides two very helpful patterns in cursor movement.

- Instead of pressing a key such as "h" or "k" a certain number of times, you can precede the command with that number. For example, **<7h>** moves the cursor seven characters to the left.
- Many lowercase commands have an uppercase equivalent that will slightly modify or enhance the command. For example, **<a>** appends text after the cursor, but **<A>** appends text after the last character at the end of the line.

The uppercase commands will be mentioned briefly in the text, and will be defined in the summary. As you try out the lowercase commands, experiment with the uppercase commands and see what they can do.

If you have not logged into the UNIX system and have not accessed **vi** to edit a file, please do so now. You will want a file that has at least 40 lines in it. If you do not have one, create one now, because you will want to try out each of these cursor movements as you read this section of the tutorial. Remember, to execute these commands, you must be in the command mode of **vi**. Press **<ESC>** to make sure you are out of the append mode, and are in the command mode of **vi**.

Character Positioning

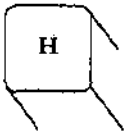
There are three ways to position the cursor by a character on a line.

- You can move the cursor right or left to a character,
- You can specify the character at either end of the line, or
- You can search for a character on a line.

Positioning the Cursor to the Right or Left

The commands, <h>, <l>, the space bar, and the BACK SPACE key move the cursor right or left to a character on the current line.

You are already familiar with the "h" and "l" keys.



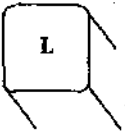
← Move the cursor to the left.

<h>

← Move the cursor one character to the left.

<nh>

Move the cursor "n" characters to the left.



→ Move the cursor to the right.

<l>

→ Move the cursor one character to the right.

<nl>

Move the cursor "n" characters to the right.

Try typing in a number before the command key. Notice that the cursor moves the specified number of characters to the left or right. In the example below, the cursor movement is depicted by the arrows.

To quickly move the cursor left or right on the screen, prefix a number to the command.

Move the cursor left 7 spaces.

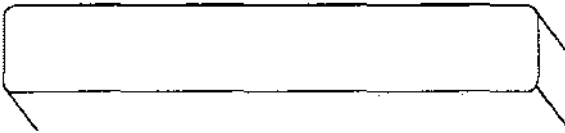
← <7h>

Move the cursor right three spaces.

<3l>→

Even if there are not 100 characters in a line, if you type in <100l>, the cursor will simply travel to the end of the line. If you type in <100h> the cursor will travel to the beginning of the line.

By now, you have probably accidentally discovered that you can move the cursor back and forth on a line using the space bar and the BACK SPACE key.



→ Space bar moves one space to the right

<space bar> → Move the cursor one character to the right.

<n space bar> Move the cursor "n" characters to the right.



Move the cursor one character to the left.

<BS>

← Move the cursor one character to the left.

<nBS>

Move the cursor "n" characters to the left.

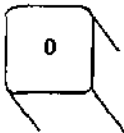
You can type in a number before the space bar or <BS>. The cursor will move that many characters to the left or right.

Positioning the Cursor at the End or Beginning of a Line

The second method of positioning the cursor on the line is shown below. These commands will place you at the first character or last character of a line.



Position the cursor on the last character of the line.



The number zero positions the cursor on the first character of the line.



The caret key positions the cursor on the first character of the line that is not a blank. (This is not a control character.)

The next examples show the movement of the cursor for each of the three commands.

Go to the back of the line!



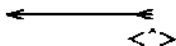
<\$>

Go to the front of the line!



<0> (The number zero)

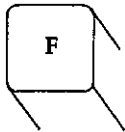
Go to the first character
of the line that
is not blank!



<^>

Searching for a Character on a Line

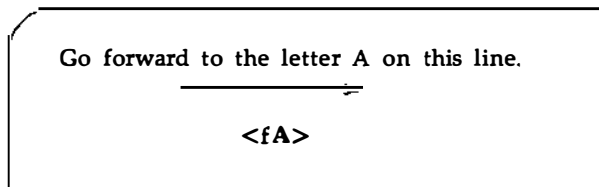
The third way to position the cursor on a line is to search for a specific character on the current line. If the character is not on the current line, a bell will sound and the cursor will not move. There is a command that will search the file for patterns. It is discussed in the next section of this tutorial.



Moves the cursor to the right to find the specified letter on the current line.

- `<fx>` \rightarrow Move the cursor to the right to the specified character *x*.
- `<Fx>` \leftarrow Move the cursor to the left to the specified character *x*.
- `<;>` The `<;>` will continue the search. It will remember the character and seek out the next occurrence of that character on the current line.

In the next example, `vi` is searching to the right for the first occurrence of the letter "A" on the current line.



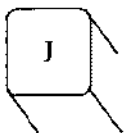
You may also find the `<tx>` command useful.

- `<tx>` \rightarrow Move the cursor to the right, to the character just before the specified character *x*.
- `<Tx>` \leftarrow Move the cursor left to the character just after the specified character *x*.

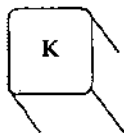
Try the search commands on one of your files. Notice the difference between the uppercase and lowercase commands.

Line Positioning

Besides the <j> and <k> commands that you have already used, the "+", "-" and RETURN keys will move the cursor line by line. The cursor will try to remain at the same position on the line. If the cursor is on the seventh character from the left in the current line, it will try to go to the seventh character on the new line. If there is no seventh character, the cursor will move to the last character.



Move the cursor down one line.



Move the cursor up on line.

Since you have already tried out <j> and <k> and know how they react, try adding a number of lines to the command as you did with <h> and <l>.

Type in: 7k

The cursor will move up seven lines above the current line. If there are not seven lines above the current line, a bell will sound and the cursor will remain on the current line.

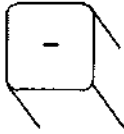
Type in: 35j

The screen will clear and redraw. The cursor will be on the 35th line below the current line. The new line will be located in the middle of the new window. If there are not 35 lines below the current line, the bell will sound and the cursor will remain on the current line. Try the following command.

Type in: 35k

Did the screen clear and redraw?

Now, try out the following three easy ways to move up or down in the file.



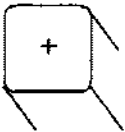
The minus sign moves the cursor up a line.

Type in: 13-

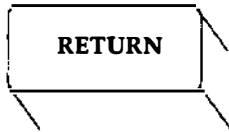
The cursor will travel up 13 lines. If some of those 13 lines are above the current window, the window will move up to reveal those lines. This is a rapid way to move quickly up the file. Try the following command.

Type in: 100-

What happened to the window? If there are less than 100 lines above the current line, a bell will sound telling you that you have made a mistake, and the cursor will remain on the current line.



or



Move the cursor down a line.

Now, try moving down the lines of the file with +.

Type in: 9+

The cursor will move down nine lines below the current line.

Try moving down line by line in the file with the RETURN key.

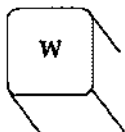
Type in: 5<CR>

Did the RETURN key give the same response as the "+" key?

Word Positioning

The vi editor considers a word a string of characters that are either numbers or letters. The word positioning commands, `<w>`, ``, and `<e>`, consider that any other character is a delimiter, telling vi it is the beginning or end of a word. Punctuation before or after a blank is considered a word. The beginning or end of a line is also a delimiter.

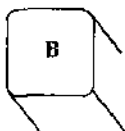
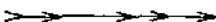
The uppercase word positioning commands, `<W>`, ``, and `<E>`, consider that the punctuation is part of the word and define a word by all the characters within two blank spaces, that is, the word is delimited by blanks.



Move the cursor to the right by words.

- `<w>` Move the cursor forward to the first character in the next word. You may press the "w" key as many times as you wish to reach the word you want, or you can prefix the number to the `<w>` command as shown below.
- `<nw>` Move the cursor forward "n" number of words to the first character of that word. The end of the line does not stop the movement of the cursor, it will wrap around and continue to count words from the beginning of the next line.
- `<W>` Ignore all punctuation, and move the cursor forward to the word after the next blank.

The **w** command
leaps word by word through the
file. Move from this word forward
<6w> _____
six words to this word.



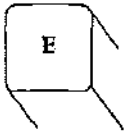
**Move the cursor backwards, to the left,
by words.**

- ** Move the cursor backward one word to the first character of that word.
- <nb>** Move the cursor backward "n" number of words to the first character of the nth word. The **** command does not stop at the beginning of a line, but moves to the end of the line above and continues to move backward.
- ** Can be used just like the **** command, except that it delimits the word only by blank spaces. It treats all other punctuation as letters of a word.

Leap backward word by word through
the file. Go back four words from here.



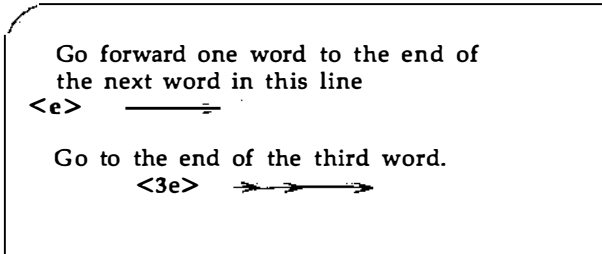
<4b>



Move forward to the end of the word.

The `<e>` command acts like `<w>` moving forward in the file by words, except that it moves the cursor to the end of the word. This makes it easy to add punctuation or add "s" to the end of a word.

The `<E>` command ignores all punctuation except blanks, delimiting the words only by blanks.



C-7

Positioning the Cursor by Sentences

The `vi` editor also recognizes sentences. In `vi`, a sentence ends in "!", or ".", or "?". If they appear in the middle of a line, they must be followed by two blank spaces for `vi` to recognize them. You should get used to the `vi` convention of putting two spaces at the end of each sentence, because you can also delete, change, or yank whole sentences, which will be discussed later in this tutorial.



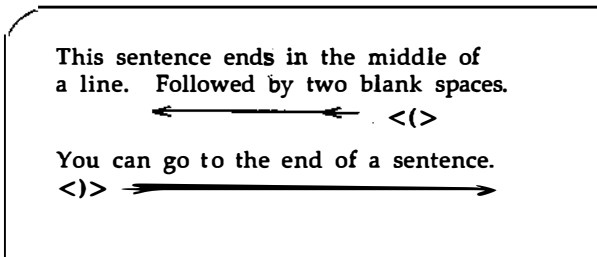
Move the cursor to the beginning of a sentence.



Move the cursor to the beginning of the next sentence.

- < (> Move the cursor to the beginning of the current sentence.
- < n(> Move the cursor to the beginning of the "nth" sentence above the current sentence.
- <) > Move the cursor to the beginning of the next sentence.
- < n) > Move the cursor to the beginning of the "nth" sentence below the current sentence.

In the next example, the arrows show the movement of the cursor.



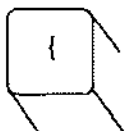
Now, precede the command with a number.

Type in: 3(or 5)

Did the cursor move the correct number of sentences?

Positioning the Cursor by Paragraphs

Paragraphs are recognized by vi if they begin after a blank line, or after the paragraph formatting command .P. If you want to be able to move the cursor to the beginning of a paragraph (or later in this tutorial, delete or change a whole paragraph), then make sure each paragraph ends in a blank line.



Move the cursor to the beginning of the current paragraph.



Move the cursor to the beginning of the next paragraph.

- < { > Move the cursor to the beginning of the current paragraph, which is delimited by a blank line above it.
- < n{ > Move the cursor to the beginning of the paragraph, "n" number of paragraphs above the current paragraph.
- < } > Move the cursor to the beginning of the next paragraph.
- < n} > Move the cursor to the "nth" paragraph below the current line.

The next example uses arrows to show the cursor moving down to the beginning of the paragraph.

The end of a paragraph is
a blank line.

This is a new paragraph.
It also ends in a blank
line. <)> →
Go to the beginning
of the next paragraph.

This is the third paragraph.

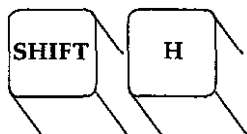
Try moving the cursor with the following commands.

Type in: {
3{
6}

Did you have enough blank lines in your file to test out the last two commands?

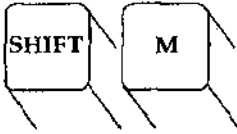
Positioning in the Window

The next three commands help you quickly position yourself in the window. Try out each of the commands.

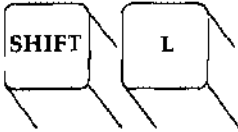


Move the cursor to the first line
on the screen.

POSITIONING THE CURSOR IN THE WINDOW



Move the cursor to the middle line on the screen.



Move the cursor to the last line on the screen.

This is the text of the file
above the current window.

This is the first line of the screen: HOME

↑
<H>

This is the MIDDLE line of the screen

↑
<M>

This is the LAST line of the screen

↑
<L>

This is the portion of text
in the file that is below the
current window.

SUMMARY OF POSITIONING IN THE WINDOW

Character Positioning Commands

- <h>** ← Move the cursor one character to the left.
- <l>** → Move the cursor one character to the right.
- <BS>** ← Move the cursor one character to the left.
- <space bar>** → Move the cursor one character to the right.
- <fx>** → Move the cursor to the right to the specified character *x*.
- <Fx>** ← Move the cursor to the left to the specified character *x*.
- <;>** Continue the search. It will remember the character and seek out the next occurrence of the character on the current line.
- <tx>** → Move the cursor to the right, to the character just before the specified character *x*.
- <Tx>** ← Move the cursor left to the character just after the specified character *x*.

Positioning by Lines

- <j>** Move the cursor down one line in the same column, if possible.

(Continued on next page)

SUMMARY OF POSITIONING IN THE WINDOW *(continued)*

- <k> Move the cursor up one line in the same column, if possible.
- <-> Move the cursor up one line.
- <+> Move the cursor down one line.
- <CR> Move the cursor down one line.

Word Positioning

- <w> Move the cursor forward to the first character in the next word.
- <W> Ignore all punctuation, and move the cursor forward to the next word delimited only by blanks.
- Move the cursor backward one word to the first character of that word.
- Move the cursor to the left one word, which is delimited only by blanks.
- <e> Move the cursor to the end of the current word.
- <E> Delimit the words by blanks only. The cursor is placed on the last character before the next blank space, or end of the line.

(Continued on next page)

SUMMARY OF POSITIONING IN THE WINDOW *(continued)*

Positioning by Sentences

< (> Move the cursor to the beginning of the current sentence.

<) > Move the cursor to the beginning of the next sentence.

Positioning by Paragraphs

< { > Move the cursor to the beginning of the current paragraph.

< } > Move the cursor to the beginning of the next paragraph.

Positioning in the Window

<H> Move the cursor to the first line on the screen, or "home".

<M> Move the cursor to the middle line on the screen.

<L> Move the cursor to the last line on the screen.

POSITIONING THE CURSOR IN THE FILE

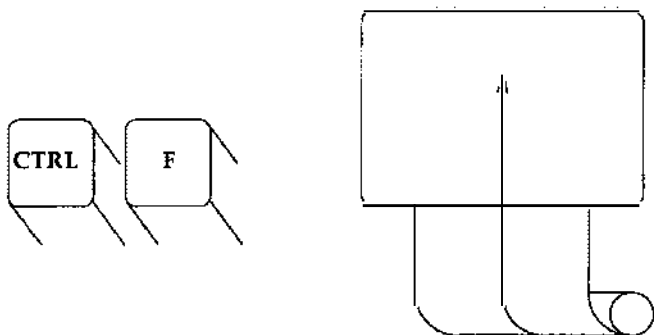
How do you move the cursor to text that is not in the current editing window? You can type in the commands <20j> or <20k>. However, if you are editing a large file, you need to move quickly

and accurately to another place in the file. This section covers those commands that help you move around within the file. You can:

- Scroll forward or backward in a file,
- Go to a specified line in the file, or
- Search for a pattern in the file.

Scrolling the Text

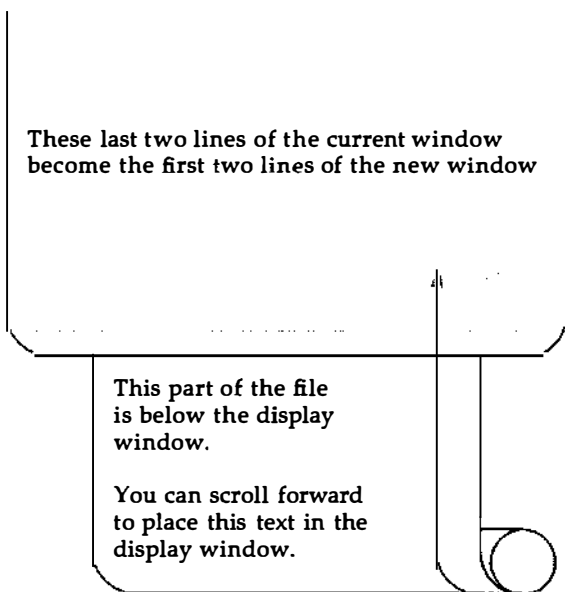
Four basic commands scroll the text of the file. `<^f>` and `<^d>` scroll the screen forward. `<^b>` and `<^u>` scroll the screen backward.



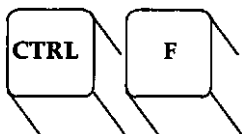
`<^f>` Scroll the text forward one full window, revealing the window of text below the current window.

To scroll the file forward, `vi` clears the screen and redraws the window. The last two lines that were at the bottom of the current window are placed at the top of the new window. If there are not enough lines left in the file to fill the window, the screen will display the `~` to indicate the empty lines.

S-1



Type in:

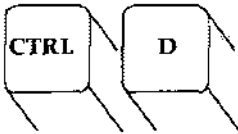


vi clears the screen and redraws the new screen shown next.

These last two lines of the current window
become the first two lines of the new window

This part of the file
is below the display
window.

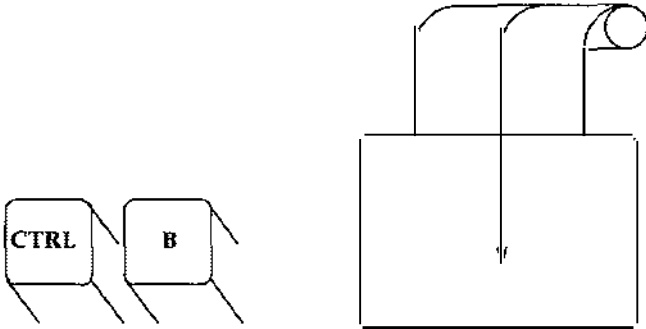
You can scroll forward
to place this text in the
display window.



**Scroll down a half screen
to reveal lines below the window.**

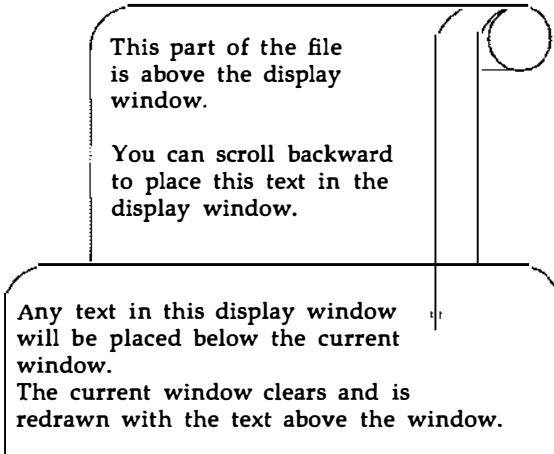
<^d> Scroll down a half screen to reveal text below the window.

When you use **<^d>**, it seems as if the text is being rolled up at the top and unrolling at the bottom to allow the lines below the screen to appear on the screen, while the lines at the top of the screen disappear. If there are not enough lines in the file, a bell will sound indicating there are no more lines.

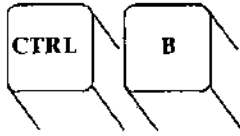


`<^b>` Scroll the screen back a full window to reveal the text above the current window.

The `<^b>` command clears the screen and redraws the window with the text that is above the current screen. Unlike the `<^f>` command, `<^b>` does not leave any reference lines from the previous window. Also, it does not use the `~` to indicate space above the top of the file. If there are not enough lines above the current window to fill a full new window, a bell will sound and the current window will remain on the screen.



Type in:



vi clears the screen and redraws the new screen shown next.

This part of the file
is above the display window.

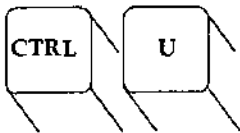
You can scroll backward
to place this text in the
display window.

Any text in this display window
will be placed below the current
window.

The current window clears and is
redrawn with the text above the window.

C-7

Any text that was in the display window is placed below the current window.



**Scroll up a half screen to reveal
lines above the window.**

<^u>

Scroll up a half window of text to reveal the lines just above the window. At the same time, the lines at the bottom of the window will be erased.

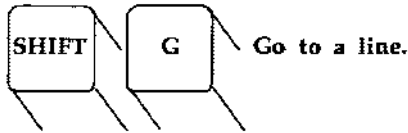
SCREEN EDITOR TUTORIAL (vi)

When you use `<^u>`, it appears as though the text in the file is on a scroll that is being unwound at the top and wound up at the bottom of the screen.

When the cursor is near the top of the file, it will move to the first line of the file and then sound a bell, alerting you it cannot scroll any farther. Try the `<^u>` and `<^d>` commands now. Watch the file scroll through the window.

Go to a Specified Line

The `<G>` command will position the cursor on a specified line in the window, or it will clear the screen and redraw the window around that line. If you do not specify a line, `<G>` will go to the last line of the file.



`<G>` Go to the last line of the file.

`<nG>` Go to the "nth" line of the file.

Line Numbers

Each line of the file has a line number, that corresponds to the number of lines in the buffer. How can you find out the line numbers? There are two basic ways. One way is to use a line editor command, which you will learn about in the section on the line editor commands. The other way is to position the cursor on the line and type in a `<^g>` command. Try the `<^g>` command now.

The `<^g>` command will give you a status notice at the bottom of the screen. The notice tells you:

- Name of the file,
- If the line has been changed [modified],
- Line number,
- Number of the last line in the file, and
- Percent the current line is of the total lines in the buffer.

This line is the 35th line of the buffer.
 The cursor is on this line.

↑ `<^g>`

There are several more lines in the
 buffer.
 The last line of the buffer is line 116.

`"file.name" [modified] line 36 of 116 --34%--`

C-7

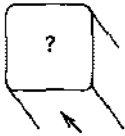
Search for a Pattern of Characters

The fastest way to reach a specific place in your text is to use one of the search commands. You can search forward or backward for the first occurrence of a specified pattern of characters or words in the buffer. The search pattern is ended by `<CR>`.

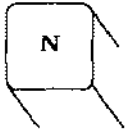
The search commands, / and ?, are not silent. They will print out on the bottom of the screen along with the search pattern. However, the command to repeat the search <n> is silent, it does not print out on the bottom of the screen.



Search forward in the buffer.



Search backward in the buffer.



Repeat the previous search.

/pattern<CR>

Search forward in the buffer for the next occurrence of the characters **pattern**. Position the cursor on the first character of the **pattern**.

/Hello world<CR>

Find the next occurrence in the buffer of the two words **Hello world**. Position the cursor under the **H**.

?pattern<CR>

Search backward in the buffer for the first occurrence of the **pattern**. Position the cursor under the first character of the **pattern**.

?data set design<CR>

Search backward in the buffer until the first occurrence of **data set design**. Position the cursor under the "d" of **data**.

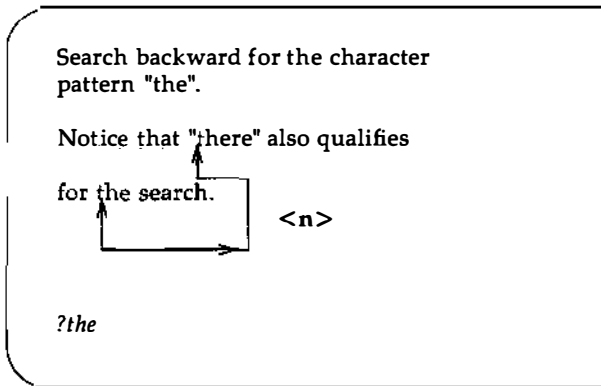
<n> Repeat the last search command.

<N> Repeat the search command in the opposite direction.

The search commands will not wrap around the end of the line in searching for two words. If you are searching for "Hello world", and "Hello" is at the end of one line, and "world" is at the beginning of another line, the search commands will not find that occurrence of "Hello world". However, the search commands will wrap around the end or the beginning of the buffer to continue the search. For example, if you are toward the end of the buffer, and the pattern you are searching for with the / command is at the top of the buffer, / will find that pattern.

The <n> command continues the last search, remembering the pattern and direction of the search.

The following example shows the results of first typing in ?the and then typing in <n>.



Experiment for a minute. What happens if you try to type in a number before ? or / or <n>? Experiment with commands in a file called *junk*. If you tried to type in a number before / or ?, you found out it does not work. However, if you tried to type in <7n>, you found out that it searched for the seventh identical pattern.

SUMMARY OF POSITIONING IN THE FILE

Scrolling

- <^f> Scroll the screen forward a full window, revealing the window of text below the current window.
- <^d> Scroll the screen down a half window, revealing lines below the current window.
- <^b> Scroll the screen back a full window, revealing the window of text above the current window.
- <^u> Scroll the screen up a half window, revealing the lines of text above the current window.

Positioning on a Numbered Line

- <G> Go to the last line of the file.
- <^g> Give the line number and status.

Searching for a Pattern

- /pattern Search forward in the buffer for the next occurrence of the **pattern**. Position the cursor on the first character of the **pattern**.

(Continued on next page)

SUMMARY OF POSITIONING IN THE FILE (*continued*)

- ?pattern** Search backward in the buffer for the first occurrence of the **pattern**. Position the cursor under the first character of the **pattern**.
- <n>** Repeat the last search command.
- <N>** Repeat the search command in the opposite direction.
-

EXERCISE 2

- 2-1. Create a file called *exer2*. Type a number on each line, numbering the lines from 1 to 50. Your files should look similar to the following.

```
1
2
3
4
5
.
.
.
45
46
47
48
49
50
```

- 2-2. Try using each of the scroll commands, notice how many lines scroll through the window. Try the following:

```
<^f>
<^b>
<^u>
<^d>
```

- 2-3. Go to the end of the file. Append the following line of text.

123456789 123456789

What number does the command **7h** place the cursor on? What number does the command **3l** place the cursor on?

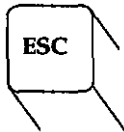
- 2-4. Try the command **\$** and the command **0** (number zero)
- 2-5. Go to the first character on the line that is not a blank. Move to the first character in the next word. Move back to the first character of the word to the left. Move to the end of the word.
- 2-6. Go to the first line of the file. Try the commands that place the cursor on the middle of the window, on the last line of the window, and on the first line of the window.
- 2-7. Search for the number 8. Find the next occurrence of number 8. Find 48.

CREATING TEXT

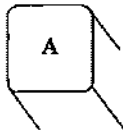
There are three basic commands for creating text:

- Append command **<a>**,
- Insert command **<i>**, and
- Open command that creates text on a new line **<o>**.

After you finish creating text with any one of these commands, you can return to the command mode of **vi** with the **<ESC>** command.

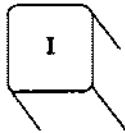


The **ESC** key ends the text input mode.

Append Text**Append text.**

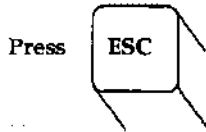
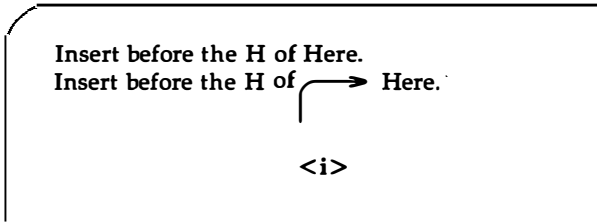
- <a>** Create text to the right of the cursor, or after the cursor.
- <A>** Append text at the end of the current line.

You have already experimented with the **<a>** command in the section on *Getting Started*. Make a new file named *junk2*. Append some text using the **<a>** command. Escape or return to the command mode of *vi* by pressing the ESC key. Then, compare the **<a>** command with the **<A>** command.

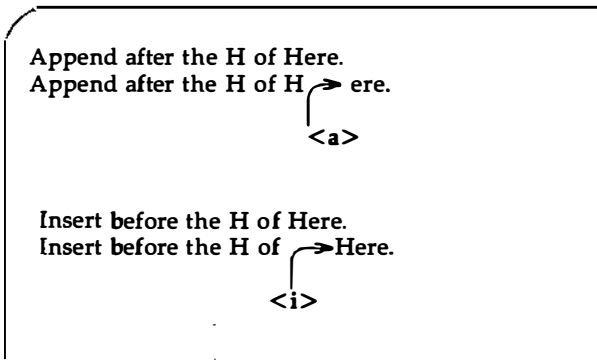
Insert Text**Insert text.**

- <i>** Insert text to the left of the cursor, or before the cursor.
- <I>** Create text at the beginning of the current line before the first character that is not a blank.

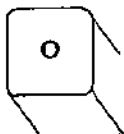
In the example below, the arrow shows where the new text will be created.



To end the insert mode and return to the command mode of vi, press the "ESC" key. In the next example you can compare the append command with the insert command.



Remember to end the append mode and the insert mode with the <ESC> command.



Create a new line of text.

<o> The open command **<o>** creates text at the beginning of a new line below the current line. The cursor can be on any character in the current line.

<O> To create text at the beginning of a new line above the current line, use the **<O>** command.

In the next screen the **<o>** command opens a new line below the current line and begins creating text at the beginning of the new line.

C-7

Create text with the open line command.

Create text below the current line.



<o>

SUMMARY OF CREATE COMMANDS

- <a> Create text after the cursor.
 - <A> Create text at the end of the current line.
 - <i> Create text in front of the cursor.
 - <I> Create text before the first character on the current line that is not a blank.
 - <o> Create text at the beginning of a new line below the current line.
 - <O> Create text at the beginning of a new line above the current line.
 - <ESC> Return vi to the command mode from any of the above text input modes.
-

EXERCISE 3

- 3-1. Create a test file *exer3*.
- 3-2. Insert the following four lines of text.

Append text
Insert text
a computer's
job is boring.

- 3-3. Create a line of text

financial statement and

above the last line.

- 3-4. Create a line of text

Delete text

above the third line using an insert command.

- 3-5. Create a line of text

byte of the budget

below the current line.

- 3-6. Using an append command create a line of text

But, it is an exciting machine.

below the last line.

- 3-7. Move to the first line and append "some" before "text".

Now, practice each of the six commands for creating text until you are familiar with using them.

- 3-8. Leave **vi** and go on to the next section to find out how to delete any mistakes you made in creating text.

DELETING TEXT

You can delete text from the text input mode or the command mode of **vi**. In addition, you can undo the effect of your most recent command that changed the buffer.

Delete Commands In the Text Input Mode

To delete text in the text input mode, you will use **<BS>**.

- <BS>** Delete the current character, the character indicated by the cursor.



Delete a character in the create mode of **vi**.

The BACK SPACE key <BS> backs up the cursor in the create mode and deletes each character that the cursor backs across. However, the deleted characters are not erased from the screen until you type over them, or use <ESC> and return to the command mode of vi.

In the next examples, the arrows show the movement of the cursor.

Press  three times.

<a>
Back space 3 spaces
←

Press 

<a>
Back space 3 spa
←

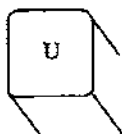
Notice that the characters do not erase from the screen until you press the ESC key.

There are two other commands that delete text in the text input mode. Although you may not use them often, you want to be aware that they are commands in the text input mode and need a special command to type them into your text, see the section on special commands.

- <^w> Delete the current word, or a specified portion of the word from the cursor to the end of the word.
- <@> Delete all of the portion of the line that is currently being created.

Undo the Last Command

Before you experiment with the commands that can delete a good portion of your text, you will want to try out the "undo" command, which will undo the last command.



Undo the last command.

- <u> Undo the last command.
- <U> Erase the last change on the current line.

If you deleted a line, <u> will bring it back on the screen. If you hit the wrong command, <u> will undo that command.

If you press the "u" key twice, it will undo the "undo". That is, if you delete a line, the first <u> will restore the line. If you press <u> again, it will delete the line again.

Delete Commands in the Command Mode

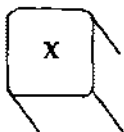
You know that you can precede a number before the command. Many of the commands in **vi**, such as the delete and change commands, allow an argument after the command. The argument can specify a text object such as a word, or a line, or a sentence, or a paragraph. The general form of a **vi** command is:

[number]command[argument]

The brackets around objects in the general form of the command line denote optional parts of the command. They are not part of the command line.

You will see many examples of this form for the delete and change commands.

All of the delete commands in the command mode of **vi** immediately remove the deleted text from the screen and redraw that part of the screen.

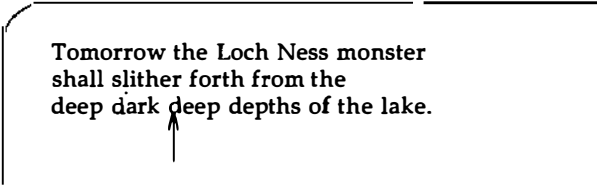


Delete a character.

<x> Delete one character.

<nx> Delete "n" characters, where n is the number of characters you want to delete.

You used **<x>** in the *Getting Started* section of this chapter. Now try preceding **<x>** with the number of characters you want to delete.

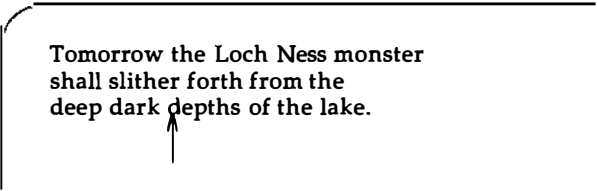


Tomorrow the Loch Ness monster
shall slither forth from the
deep dark deep depths of the lake.

Put the cursor on the first letter you want to delete, in this example the "d" of the second "deep".

Type in: 5x

The screen will delete "deep", plus the extra space, and readjust the text on the screen so that it will now read:



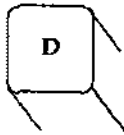
Tomorrow the Loch Ness monster
shall slither forth from the
deep dark depths of the lake.

You can also use the delete word command, which is discussed next.

Delete Text Objects

The delete command follows the general form of a vi command.

[number]d[text object]

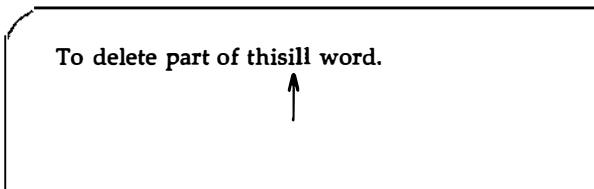


Delete a word, a line, a sentence, or a paragraph.

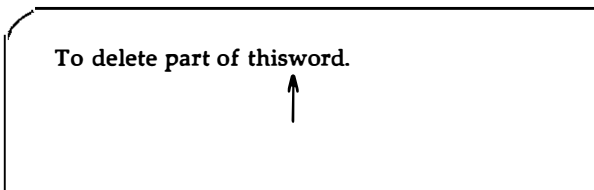


Delete a word.

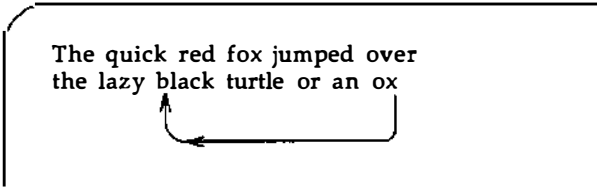
You can delete all of a word or part of a word with `<dw>` by moving the cursor to the first character you want deleted. Pressing `<dw>` deletes that character and all characters up to and including the next space or punctuation character.



Type in: `dw`

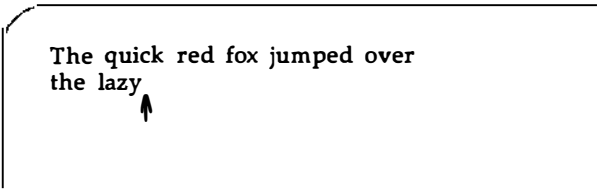


You can delete one word with `<dw>` or several words by prefixing the "dw" with a number. The cursor must be on the first character of the first word to be deleted. To delete five words, you would type in `5dw`. An example of how to do this follows.



The quick red fox jumped over
the lazy black turtle or an ox

Type in: **5dw**

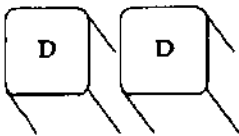


The quick red fox jumped over
the lazy

Try typing in the arguments for other text objects that you learned in the section on positioning the cursor.

Type in: **d(or d)**

Observe what happens to your file. Remember, you can restore the text that you just deleted with **<u>**.



<dd> Delete a line of text.

To delete a line, press the "d" key twice. You do not need to worry about deleting text if you press the "d" key once. Nothing will

happen, unless you press the space bar. The **<d space bar>** acts like the **<x>** command and deletes one character. If you accidentally press "d" key in the command mode, press the ESC key. The ESC key will cancel the previous typed command.

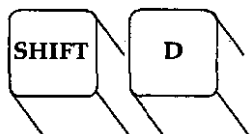
Try to delete ten lines.

Type in: **10dd**

The lines will be deleted from the screen. If some of the lines are below the current window, **vi** will display a notice on the bottom of the screen:

10 lines deleted

If there are not ten lines below the current line in the file, a bell will sound and no lines will be deleted.



Delete the line from the cursor to the end of the line.

If you are erasing the end of a line, use the **<D>** command. Put the cursor on the first character to be deleted, hold down the **SHIFT** key while you press the "d" key.

Type in: **D**

The **<D>** command will not allow you to specify more than the current line. You cannot type in "3D". However, you could type in **<3d\$>**. Remember the general form of a **vi** command? The **\$** refers to the end of the line in **vi**.

SUMMARY OF DELETE COMMANDS

For the CREATE Mode:

- <BS>** Delete the current character.
- <^h>** Delete the current character.
- <^W>** Delete the current word.
- <@>** Delete the current line of new text, or delete all new text on the current line.

For the COMMAND Mode:

- <u>** Undo the last command.
- <U>** Erase the last change on the current line.
- <x>** Delete the current character.
- <ndx>** Delete "n" number of text objects "x".
- <dw>** Delete the word at cursor through the next space or to the next punctuation mark.
- <dd>** Delete the current line.
- <D>** Delete the line at the cursor to the end of the line.
- <d>** Delete the current sentence.
- <d|>** Delete the current paragraph.

EXERCISE 4

- 4-1. Create a file *exer4* containing the following four lines:

When in the course of human events
there are many repetitive, boring
chores, then one ought to get a
robot to perform those chores.

- 4-2. Move the cursor to line 2 and append to the end of that line:

tedious and unsavory.

Delete "unsavory" while in the append mode.

Delete "boring" in the command mode.

What is another way you could have deleted "boring"?

- 4-3. Insert at the beginning of line 4:

congenial and computerized.

Delete the line.

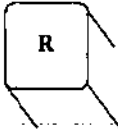
How could you delete the line and leave it blank?

Delete all the lines with one command.

- 4-4. Leave the screen editor and remove the empty file from your directory.

CHANGING TEXT

Instead of deleting text using a delete command and then creating text with a text input command, the three basic commands, `<r>`, `<s>`, and `<c>` both erase the text and then create new text.

Replacing Text

Replace one character that is typed over.

- <r>** Replace the current character, the character pointed to by the cursor. This is not a text input mode. It does not need to be ended by <ESC>.
- <nr>** Replace "n" characters with the same letter. This command automatically terminates after "nth" character is replaced. It does not need the <ESC>.
- <R>** Replace only those characters typed over until the <ESC> command is given. If the end of the line is reached, this command will then begin appending new text.

The <r> command will replace the current character with the next character that is typed in. For example, in the sentence below you want to change "acts" to "ants".

The circus has many acts.

Place the cursor under the "c" of "acts".

Type in: rn

The sentence becomes:

The circus has many ants.

To change "many" to "6666", place the cursor under the "m" of "many".

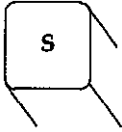
Type in: 4r6

The <r> command changes the four letters of "many" to 6s.

The circus has 6666 ants.

Substituting Text

The substitute command replaces characters, but then allows you to continue to create text from that point until you press <ESC>.



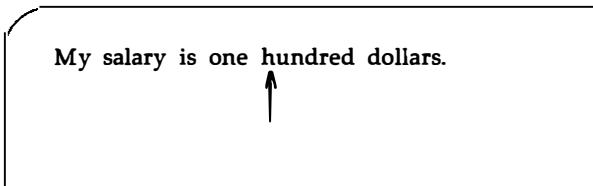
Substitute for a character of text.

- <s> Delete the character the cursor is on and append text. End the text input mode with the ESC key.
- <ns> Delete "n" characters and append text. End the text input mode with <ESC>.
- <S> Replace all the characters in the line.

The <s> command indicates the last character in the substitution with a \$. The characters are not erased from the screen until you type over them, or leave the text input mode with the <ESC> command.

Notice that you cannot use an argument with either <r> or <s>. Did you try?

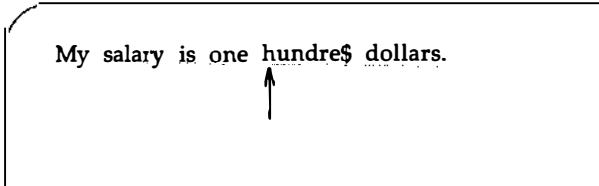
Suppose you want to substitute "million" for "hundred" in the following example.



Put the cursor under the h of hundred.

Then type in: 7s

Notice where the \$ is placed.



My salary is one hundres\$ dollars.

Now type in: million

Press the ESC key, and you will owe the Internal Revenue Service \$500,000.

Changing Text

The substitute command replaces characters. The change command replaces text objects, and then continues to append text from that point until you press <ESC>. To end the change command and return to the command mode in vi, you must press the ESC key.



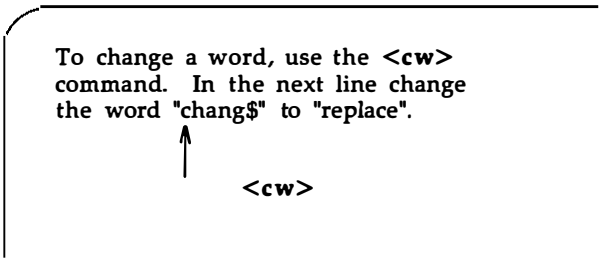
Change. Replace a text object with new text.

The change command can take an argument. You can replace a character, word, or an entire line with new text.

- <cw> Replace a word or the remaining characters in a word with new text. The vi editor prints a \$ indicating the last character to be changed.
- <ncw> Replace "n" number of words with new text.

- <cc>** Replace all the characters in the line.
- <ncc>** Replace all the characters in the current line and up to "n" lines of text.
- <ricx>** Replace "n" number of text objects "x", such as sentences) and paragraphs }.
- <C>** Replace the remaining characters in the line, from the cursor to the end of the line.
- <nC>** Replace the remaining characters from the cursor in the current line and replace all the lines under the current line up to "n" lines.

For the **<cw>** command and the **<C>**, a **\$** will indicate the last letter that will be replaced. The characters will remain on the screen until you have pressed the ESC key. When used to change one or more lines of text, the change command simply deletes the lines that are to be replaced, and then places you in the text input mode of vi.



In the example, notice that "replace" has more letters than "change". Once you have executed the change command you are in the text input mode of vi and you can add as much text as you want, until you press **<ESC>**.

To change a word, use the `<cw>` command. In the next line change the word "replace" to "replace".



`<ESC>`

Try the other change commands. Watch the screen. When you use `<C>` the \$ will appear at the end of the line. Try using other arguments.

Type in: `cl`

Since you know the undo command, do not hesitate to experiment with different arguments, or preceding the command with a number. You must press `<ESC>` before you can use `<u>` since `<c>` places you in a text input mode.

Compare `<S>` to `<cc>`. The results should be the same for both commands.

SUMMARY OF CHANGE COMMANDS

- `<r>` Replace only the current character.
- `<R>` Replace only those characters typed over with new characters until the `<ESC>` command is given.
- `<s>` Delete the character the cursor is on and append text. End the append mode with the ESC key.
- `<S>` Replace all the characters in the line.

(Continued on next page)

SUMMARY OF CHANGE COMMANDS *(continued)*

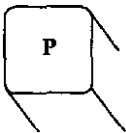
- <cw> Replace a word or the remaining characters in a word with new text.
 - <cc> Replace all the characters in the line.
 - <ncx> Replace "n" number of text objects "x", such as sentences) and paragraphs }.
 - <C> Replace the remaining characters in the line, from the cursor to the end of the line.
-

CUTTING AND PASTING TEXT ELECTRONICALLY

There is a set of commands that will cut and paste text in a file. Another set of commands will copy a portion of text and place it in another section of a file.

Moving Text

You can move text from one place to another in the vi buffer by deleting the lines and then placing them at the spot in the text that you want them. The last text or lines that were deleted go into a temporary buffer. If you move the cursor to that part of the file where you want the deleted lines to be placed and press the "p" key, the deleted lines will be added below the current line.



The put command <p> puts the last yank or delete in the proper place.

- <p> Place the contents of the temporary buffer after the cursor.
- <np> Place "n" number of copies of the temporary buffer after the cursor.

A partial sentence that was deleted by the <D> command can be placed in the middle of another line. Position the cursor in the space between two words, then press "p". The partial line is placed after the cursor.

Characters deleted by <nx> also go into a temporary buffer. Any text object that was just deleted can be placed somewhere else in the text with <p>.

The <p> command should be used right after a delete command since the temporary buffer only stores the results of one command at a time. The <p> command also places a copy of text after the cursor that had been placed in the temporary buffer by the yank command. Yank <y> is discussed next in *Copying Text*.

Fixing Typos

A quick way to fix typos that consist of transposed letters is to combine the <x> and the <p> commands as <xp>. <x> deletes the letter. <p> places it after next character.

Notice the error in the next line.

A line of tetx

This error can be quickly changed by placing the cursor under the "t" in "tx" and then pressing first "x" and then "p" keys. The result is:

A line of text

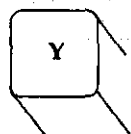
Try it. Make a typing error in your file. Then use <xp>.

Copying Text

You can "yank" (copy) a part of the text into a temporary buffer, then move the cursor to that part of the file where you want to place a copy of the text, and place it there. `<p>` places the text after the current line.

The "yank" command follows the general form of a `vi` command. It allows you to specify the number of text objects that you want copied.

[number]y[text object]



The "yank" command `<y>` saves a copy of the text object.

- `<yw>` Yank a copy of a word.
- `<yy>` Yank a copy of the current line into a temporary buffer to be placed below another line.
- `<nny>` Yank "n" lines into a temporary buffer to be placed below the current line. "n" is the number of lines.
- `<y>` Yank a copy of a sentence.
- `<y}>` Yank a copy of the paragraph.
- `<nxy>` Yank "n" number of text objects "x", such as sentences) and paragraphs }.

Try the following command lines and see what happened to your screen. Of course you can undo the last command.

Type in: `5yw`

Move the cursor to another spot.

Type in: `p`

Try yanking a paragraph <y]> and placing it after the current paragraph, then move to the end of the file <G> and place that same paragraph at the end of the file.

Copying or Moving Text Using Registers

If you have several sections of text that you wish moved or copied to a different part of the file, it would be tedious to move each portion one at a time. vi has named registers, which are electronic storage boxes where you can store the text until you want to place it into a specific spot in the file. These registers are named for each letter of the alphabet, a through z. You can either yank or delete text to one of these registers.

These commands are handy if you have an example that you want to use several times in the text. The example will stay in the specified register until you end the editing session or yank or delete another section of text to that register.

The general form of the command is:

[number]lcommand[text object]

The l represents any letter, and is the name of the register. You can precede the command with a number to indicate how many text objects, such as words or lines, that you want to save in the register.

Place the cursor at the beginning of a line.

Type in: 3"ayy

Now, type in more text. Then, go to the end of the file.

Type in: "ap

Did the lines you saved in register "a" appear at the end of the file?

SUMMARY OF CUT AND PASTE COMMANDS

- <p> Place the contents of the temporary buffer containing the last delete or yank command into the text after the cursor.
 - <yy> Yank a line of text and place it into a temporary buffer.
 - <nrx> Yank a copy of "n" number of text objects "x" and place them in a temporary buffer.
 - <"lyn> Place a copy of text object "n" in the register named by a letter "l".
 - <"lp> Place the contents of register l after the cursor.
-

EXERCISE 5

- 5-1. Edit the file *exer2*. Notice that this is the same file you created in Exercise 2.

Go to line 8 and change that line to read "END OF FILE".

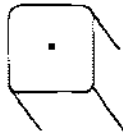
- 5-2. Yank the first eight lines of the file and place them in register "z". Put the contents of register "z" after the last line of the file.
- 5-3. Go to line 8 and change that line to read "8 is great".
- 5-4. Go to line 18 and make the same change as you did in 5-3.
- 5-5. Go to the last line of the file. Substitute "EXERCISE" for "FILE". Replace "OF" with "TO".

SPECIAL COMMANDS

There are some special commands that you will find useful.

- <.> Repeat the last command.
- <J> Join two lines together.
- <\>
or
<^v> Print out nonprinting character.
- <^l> Clear the screen and redraw it.
- <~> Change lowercase to uppercase and vice versa.

Repeating the Last Command



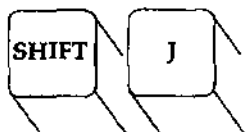
Repeat the last
change command.

You may have already accidentally pressed the "." key, thinking that you were adding a period at the end of your sentence. If you were in the command mode of vi, you were unpleasantly surprised by the last text change suddenly appearing on the screen.

The period repeats the last change command. This is a very handy command when it is used with the search command. For example, you forgot to capitalize the "S" in United States. However, you do not want to capitalize the "s" in "chemical states". One way you could correct this problem is search for "states". The first time you found "states" in United states, you would change the "s" to "S". The next occurrence you found, you would simply press the "." key and vi would remember to change the "s" to "S".

The <.> will repeat change, or create, or delete, or put commands. Experiment with the commands. Watch the screen to see how the text is affected.

Joining Two Lines



Join the line below the current line with the current line.

The `<J>` command joins lines. Place the cursor on the current line, hold down the `SHIFT` key and press the "j" key. The line below the current line is joined to the current line at the end of the current line.

Now is the time to join forces.

To join these two lines into one line, place the cursor under any character in the first line.

Type in: `J`

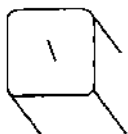
Those two lines become:

Now is the time to join forces.

Notice that `vi` automatically places a space between the last word on the first line and the first word on the second line.

Typing Nonprinting Characters

In the section of this tutorial on deleting in the text input mode, two commands were mentioned that are probably seldom used, but act as commands and will not print out in your text. How do you get characters that are commands in the text input mode to type out in your text? Precede them with a `\`.

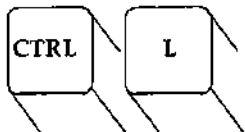


Type in nonprinting characters.

What happens when you want to type in the @ character? Try it. It erased the line you are working on. How do you type in the @ character?

Type in: \@

Clearing and Redrawing the Window



Clear and redraw the current screen.

One of the frustrating things that can happen to you in **vi** is that another user in your UNIX system decides to send you a message using the **write** command. If you have not turned off your messages in the shell, the message will appear right at the spot where you are editing in the current window. After you have read the message, how do you restore the current window? If you are in the text input mode, you must end it with the <ESC> command to get you into the command mode of **vi**. Then, hold down the CTRL key and press the "l" key. **vi** will clear away the garbage, and redraw the window exactly as it was before the message arrived.

Changing Lowercase to Uppercase and Vice Versa



Change uppercase to lowercase,
or lowercase to uppercase.

A quick way to change any lowercase letter to a capital letter or any capital letter to lowercase is the `<~>` command. To change a to A, or B to b press `~`. This command does not allow you type in a number before the command and change several letters with one command.

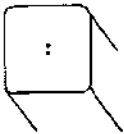
SUMMARY OF SPECIAL COMMANDS

- `<.>` Repeat the last command.
 - `<J>` Join the line below the current line with the current line.
 - `<\x>` Print the nonprinting character `x` that does not print out in the text input mode.
 - `<^v>` Print characters that do not normally print out in the text input mode.
 - `<^I>` Clear and redraw the current window.
 - `<~>` Change lowercase to uppercase, or vice versa.
-

LINE EDITING COMMANDS

The screen editor `vi` also has some line editing capabilities. The line editor associated with `vi` is called `ex`. However, the `ex` commands are very similar to the `ed` commands. If you know the `ed` commands, you may want to experiment on a test file and see how many will work in `vi`.

There are many commands in the `ex` editor that can be called from `vi`. Only a few of the most useful commands are discussed here.



Call in the line editor commands.

To call in the line editor commands, type in a ":" from the command mode of **vi**. The cursor will drop down to the bottom of the screen and display the ":". As you try out the line editing commands notice that they print out at the bottom of the editing window.

A powerful and useful command of **ex** is the command that temporarily returns you to the shell. You can return to the shell, perform some shell commands (even edit and write another file in **vi**) and then return to the current window of **vi**.

:sh<CR> Temporarily return to the shell, leaving the **vi** buffer with the cursor on the current line.

<^d> After you have executed the shell commands, hold CTRL and press "d". You will return to the exact line and window you were editing before you left **vi**.

Even if you change directories while you are temporarily in the shell and then execute **<^d>**, you will return to the **vi** buffer in the directory where you were editing the file.

Write Text to a New File

What do you do if you want only part of the file in the editing buffer placed in a UNIX system file?

Many of the commands in **ex** will accept a line number or a range of line numbers typed in before the command **w**. Try to write the third line of the buffer to a file named *three*.

Type in: **:3w three<CR>**

Notice the system response.

"three" [New file] 1 line, 20 characters

The "." is the special character that indicates the number of the current line.

Type in: `:.w junk<CR>`

A new file called *junk* will be created containing only the current line in the *vi* buffer.

You can also specify the range of lines. To write lines 23 through 37 to a file, type in:

`23,37w newfile<CR>`

Finding the Line Number

If you want to specify a range of lines, you can find out the line number of that line by moving the cursor to that line.

Type in: `:=<CR>`

The editor will come back with the response that is the number of that line.

If you want to know the number
of this line, type in `:=<CR>`

`:=`

As soon as you press RETURN, the bottom line will clear and give you the number of the line in the buffer.

If you want to know the number
of this line, type in `:=<CR>`

34

You can move the cursor to any line in the buffer by typing in a ":" and the line number.

`:n<CR>` Go to the "nth" line of the buffer.

Deleting the Rest of the Buffer

One of the easiest ways to delete all the lines from the current line to the end of the buffer is to use the line editor command to delete lines.

Type in: `.,,$d<CR>`

The "." is the current line, and the last line is \$.

Adding a File to the Buffer

If you have a file with some data or text in it that you would like to add below a specific line in the editing buffer, you can do so with the `:r` command. To read in the file *data* place the cursor on the line above the desired insertion.

Type in: `:r data<CR>`

You may also specify the line number instead of moving the cursor. Insert the file *data* below line 56 of the buffer.

Type in: `:56r data<CR>`

Do not be afraid to experiment, `<u>` will undo the `ex` commands too.

Making Global Changes

One of the most powerful commands in `ex` is the global command. The global command is given here to help those users who are familiar with the line editor. Even if you are not familiar with a line editor, you may want to try the command on a test file.

If you had typed in several pages of text about the DNA molecule, calling its structure a "helix", you would have to change each occurrence of the word "helix" to "double helix". This could be a long

involved process searching for each one and probably using the "." command of **vi** to repeat the change. If you are sure you want every "helix" changed, you can use the global command of **ex**. You need to understand a series of commands to do this. Let's take one at a time.

:g/characters<CR>

Search for these exact characters.

Type in: **:g/helix<CR>**

The line editor does a global search for the first instance of the characters "helix" on a line.

:s/text/new words/<CR>

This is the substitute command. Instead of writing over the word **text**, as the screen editor would have done, the line editor searches for the first instance of the characters **text** on the current line, and changes them to **new words**. You must tell **ex** what word you are looking for and it must appear between the first two delimiters, /. It will then replace only those exact characters with the exact characters, **new words**, between the last two delimiters.

:s/text/new words/g<CR>

By adding a "g" at the end of the last delimiter of this command line, **ex** will change every occurrence on the current line.

:g/helix/s//double helix/g<CR>

This command line searches for the word **helix**. Each time **helix** is found, the substitute command substitutes **double helix** for every instance of **helix** on that line. The delimiters after the **s** do not need to have **helix** typed in again. The command remembers the word from the delimiters after the global command **g**.

This is a very powerful command. If it is confusing to you, but you still want to add it to your vi command knowledge, read *Chapter 5* on the line editor *ed* for a more detailed explanation of the global and substitution commands.

SUMMARY OF LINE EDITOR COMMANDS

:	Indicates that the next commands are line editor commands.
:sh<CR>	Temporarily return to the shell to perform some shell commands.
<^d>	Escape the temporary shell and return to edit the current window of vi.
:n<CR>	Go to the "nth" line of the buffer.
:x,zw data<CR>	Write lines from the number "x" through the number "z" into a new file called <i>data</i> .
:\$<CR>	Go to the last line of the buffer.
:\$d<CR>	Delete all the lines in the buffer from the current line to the last line.
:r shell.file<CR>	Insert the contents of <i>shell.file</i> under the current line of the buffer.
:s/text/new words/<CR>	Replace the first instance of the characters text on the current line with new words .
:s/text/new words/g<CR>	Replace every occurrence of text on the current line with new word .
:g/text/s//new word/g<CR>	Change every occurrence of text to new word .

QUITTING VI

There are six basic command sequences to quit the **vi** editor.

<ZZ> Write the contents of the **vi** buffer to the UNIX system file currently being edited and quit **vi**.

:wq<CR> Write the contents of the **vi** buffer to the UNIX system file currently being edited and quit **vi**.

:w filename<CR>

:q<CR> Write the temporary buffer to a new file named *filename* and quit **vi**.

:w! filename<CR>

:q<CR> Overwrite an existing file called *filename* with the contents of the buffer and quit **vi**.

:q!<CR> Quit **vi** without writing to the shell file.

:q<CR> Quit **vi** without writing the buffer to a UNIX system file. This command, without the write command **w**, can only be used in special cases, such as the **view** command discussed in the next section, or if the buffer has not been changed.

The commands that are preceded by a ":" are line editor commands.

The **<ZZ>** command and **:wq** command sequence both write the buffer to a UNIX system file, then quit **vi**, and return you to the shell command level. You have tried the **<ZZ>** command, now try to exit **vi** with **:wq**.

Type in: **:wq<CR>**

The system response is the same as it is for the **<ZZ>** command. It gives you the name of the file, and the number of lines and characters in the file.

vi remembers the file name of the file currently being edited, so you do not have to reiterate the file name when you want to write the buffer of the editor back into that file. What do you do if you want to give the file a different name?

If you want to write to a file called *junk*:

Type in: `:w junk<CR>`

After you write to a new file, you can leave *vi* by just typing in the `:q`.

Type in: `:q<CR>`

If you try to write to a file called *letter* that already exists in the shell, you will receive a warning:

"letter" File exists - use "w! letter" to overwrite

Type in: `:w! letter<CR>`

You will erase the current file called *letter* and overwrite it with the new file.

If you began editing a file called *memo*, made some changes to the file, and then decided you didn't want to make the changes, or you accidentally pressed a key that gave *vi* a command you could not undo, you can leave *vi* without writing to the file.

Type in: `:q!<CR>`

SUMMARY OF QUIT COMMANDS

`<ZZ>`

Write the file and quit *vi*.

`:wq<CR>`

Write the file and quit *vi*.

`:w filename<CR>`

`:q<CR>`

Write the editing buffer to a new file named *filename* and quits *vi*.

(Continued on next page)

SUMMARY OF QUIT COMMANDS (*continued*)

- | | |
|--|---|
| :w! filename<CR>
:q<CR> | Overwrite an existing file called <i>filename</i> with the contents of the editing buffer and quits vi . |
| :q!<CR> | Quit vi without writing to the buffer. |
| :q<CR> | Quit vi without writing the buffer to a UNIX system file. |
-

SPECIAL OPTIONS FOR vi

The **vi** command has some special options. It allows you to:

- Recover a file lost by an interrupt to the UNIX system,
- Place several files in the editing buffer and edit each in sequence, and
- View the file with the **vi** cursor positioning commands.

Recovering a File Lost by an Interrupt

There are times when an interrupt or a disconnect will cause the system to exit the **vi** command without writing the temporary buffer to the UNIX system file. Or, you may become confused or have a problem with the **vi** editor that you cannot solve. If that happens, one solution is simply to hang up, or disconnect from the UNIX system. In both of these cases, the UNIX system will store a copy of the buffer for you. When you log back into the UNIX system you will want to restore the file with the **-r** option for the **vi** command:

Type in: **vi -r filename<CR>**

The changes you made to the file *filename*, before the interrupt occurred, are now in the vi buffer. You can continue editing the file, or you can write the file and quit vi. The vi editor will remember the file name and write to that file.

Editing Multiple Files

If you wish to edit more than one file in the same editing session, type in the vi command followed by each file name.

Type in: `vi file1 file2<CR>`

vi will respond by telling you how many files you are going to edit.

2 files to edit

After you have edited the first file, *file1*, you need to write the changes to the shell file.

Type in: `:w<CR>`

The system response to the `:w <CR>` command will be a message at the bottom of the screen giving the name of the file, and how many lines and characters are in that edited file. Then you must ask for the next file in the editing buffer with the `:n` command.

Type in: `:n<CR>`

The system response to the command `:n<CR>` is a notice at the bottom of the screen with the name of the next file to be edited and the character and line count of that file.

Pick two of the files in your current directory and enter vi to place the two files in the editing buffer at the same time. Notice the system responses to the commands at the bottom of the screen.

SUMMARY OF SPECIAL OPTIONS FOR vi

- vi file1 file2 file3<CR>** Enter three files into the vi buffer to be edited. Those files are *file1*, *file2*, and *file3*.
- :w<CR>**
:n<CR> Write the current file and call the next file in the buffer.
- vi -r file1<CR>** Restore the changes made to the file *file1*.
-

EXERCISE 6

- 6-1. Try to restore a file lost by an interrupt.

Enter vi, create some text in a file called *exer6*.

Turn off your terminal without writing to a file or leaving vi.

Log back in to your terminal.

Try to get back into vi and edit the *exer6* file.

- 6-2. Place *exer1* and *exer2* in the vi buffer to be edited.

Write *exer1* and call in the next file in the buffer, *exer2*.

Write *exer2* to a file called *junk*.

Quit vi.

- 6-3. Try out the command:

vi exer*<CR>

What happens? To quit vi:

Type in: **ZZ ZZ**

6-4. Look at *exer4* in read only mode.

Scroll forward.

Scroll down.

Scroll backward.

Scroll up.

Quit and return to the shell.

CHANGING YOUR ENVIRONMENT

If you are going to edit with **vi** you will want to change your login environment so that you do not have to reconfigure your terminal each time you login. Your login environment is controlled by a file in your login directory called the *.profile*. The *.profile* is explained in more detail in the shell tutorial in *Chapter 7*.

You are about to edit your *.profile* that sets up your environment each time you login. If you are concerned that you might cause a problem with your *.profile* in the editing process, you may want to keep a backup copy of your original *.profile* for safekeeping.

From your login directory, type in:

```
cp .profile safe.profile<CR>
```

Now that you have a copy of your *.profile* in a safe place, *safe.profile*, you can edit your *.profile* just like any other file in **vi**.

Type in: **vi .profile<CR>**

Go to the last line of the file, ignoring all the lines currently in the file.

Type in: **G**

You are going to add two lines to the bottom of the file, the same terminal configuration you typed in at the beginning of your login session so that you could enter **vi**.

Type in: `<o>`

Now you are ready to append text to the end of the file.

Type in: `TERM=code<CR>`
`export TERM<CR>`

Remember "code" is the special code characters for your type of terminal.

Write and quit **vi**. Now, the next time that you log into the UNIX system **TERM** is automatically set and you can immediately begin editing with **vi**.

Setting the Automatic Carriage Return

If you want an automatic carriage return, create a new file `.exrc`. The `.exrc` file controls the editing environment for **vi**. There are several options you can set in this file. If you want to know more about `.exrc`, read the *Editing Guide*. (See *Appendix A*.)

Type in: `vi .exrc<CR>`

Add one line to this file.

Type in: `wm=n<CR>`

"n" is the number of characters from the right side of the screen where the carriage return will occur. If you want a carriage return at 20 characters from the right edge of the screen,

Type in: `wm=20<CR>`

Write and quit that file. The next time you login this file will give you an automatic carriage return.

You can check on these settings, the terminal setting and the wrapmargin (automatic carriage return) when you are in **vi**.

Type in: `:set<CR>`

vi will tell you the terminal type and the wrapmargin. You can also use the **:set** command to create or change the wrapmargin. Try experimenting with it.

Now you know the basics of **vi**! Experiment with the commands, find the ones that work best for you.

C-7

ANSWERS TO EXERCISES

There is often more than one way to perform a task in vi. If the way you tried worked, then your answer is correct. Below are suggestions for performing the task given in the exercise.

Exercise 1

1-1. Look up your terminal code with the following command. Type in:

```
grep "your type of terminal" /etc/termcap<CR>
```

The first two letters of of the system response are your terminal code.
Type in:

```
TERM=code<CR>  
export TERM<CR>
```

1-2. Type in:

```
vi exer1<CR>  
<a>  
This is an exercise!<CR>  
Up, down<CR>  
left, right,<CR>  
build your terminal's<CR>  
muscles bit by bit.<ESC>
```

1-3. Use the <k> and the <h> commands.

1-4. Use <x>.

1-5. Use the <j> and <l> commands.

1-6. Type in:

```
<a> <CR>  
and byte by byte<ESC>
```

Use <j> and <l> to move to the last line and character of the file.
Use <a> to add text. <CR> will create the new line. <ESC> will
end the create mode.

1-7. Type in:

```
ZZ
```

1-8. Type in:

```
vi exer1<CR>
```

System response:

"exer1" 6 lines, 100 characters

Exercise 2

2-1. Type in:

```
vi exer2<CR>
<a>1<CR>
2<CR>
3<CR>
.
.
.
48<CR>
49<CR>
50<ESC>
```

2-2. Type in:

```
<^f>
<^b>
<^u>
<^d>
```

Notice the line numbers as the screen changes.

2-3. Type in:

```
<G>
<o>
123456789 123456789<ESC>
```

2-4. \$ = end of line

0 = first character in the line

2-5. Type in:

```
<^>
<w>
<b>
<e>
```

2-6. Type in:

```
<1G>
<M>
<L>
<H>
```

2-7. Type in:

```
/8  
<n>  
/48
```

Exercise 3

3-1. Type in:

```
vi exer3<CR>
```

3-2. Type in:

```
<a> Append text <CR>  
Insert text<CR>  
a computer's <CR>  
job is boring.<ESC>
```

3-3. Type in:

```
<O>  
financial statement and<ESC>
```

3-4. Type in:

```
<3G>  
<i>Delete text<CR> <ESC>
```

The text in your file now reads:

```
Append text  
Insert text  
Delete text  
a computer's  
financial statement and  
job is boring.
```

3-5. The current line is "a computer's". To create a line of text below that line use the <o> command.

3-6. The current line is "byte of the budget".

<G> will put you on the bottom line.

<A> will begin appending at the end of the line.

<CR> will create the new line.

Then, type in the text "But, it is an exciting machine."

<ESC> ends the append mode.

3-7. Type in:

```
<1G>
/text
<i>some<space bar> <ESC>
```

3-9. <ZZ> will write the buffer to *exer3* and put you in the command mode of the shell.

Exercise 4

4-1. Type in:

```
vi exer4<CR>
<a> When in the course of human events<CR>
there are many repetitive, boring<CR>
chores, then one ought to get a<CR>
robot to perform those chores.<ESC>
```

4-2. Type in:

```
<2G>
<A> tedious and unsavory<CR>
<8BS>
<ESC>
```

Press <h> until you get to the "b" of "boring" then press <dw>. Or, you could have used <6x>.

4-3. You are at the second line. Type in:

```
<2j>
<I> congenial and computerized<ESC>
<dd>
```

To delete the line and leave it blank, type in:

```
<0> (zero to place you at the beginning of the line)
<D>

<H>
<3dd>
```

4-4. Write and quit vi.

```
<ZZ>
```

Remove the file.

```
rm exer4<CR>
```

Exercise 5

5-1. Type in:

```
vi exer2<CR>
<8G>
<cc> END OF FILE <ESC>
```

5-2. Type in:

```
<1G>
<8"zyy>
<G>
<"zp>
```

5-3. Type in:

```
<8G>
<cc> 8 is great<ESC>
```

5-4. Type in:

```
<18G>
<.>
```

5-5. Type in:

```
</FI>
<cw> EXERCISE<ESC>

<?OF>
<R>TO<ESC>
```

Exercise 6

6-1. Type in:

```
vi exer6<CR>
<a> (append several lines of text)
<ESC>
```

Turn off the terminal.

Turn on the terminal.

Log into the UNIX system. Type in:

```
vi -r exer6<CR>
:wq<CR>
```

6-2. Type in:

```
vi exer1 exer2<CR>
:w<CR>
:n<CR>
```

```
:w junk<CR>
ZZ
```

6-3. Type in:

```
vi exer*<CR>
```

(Response)

*8 files to edit (vi calls in all files with
names that begin with exer.)*

```
ZZ
ZZ
```

6-4. Type in:

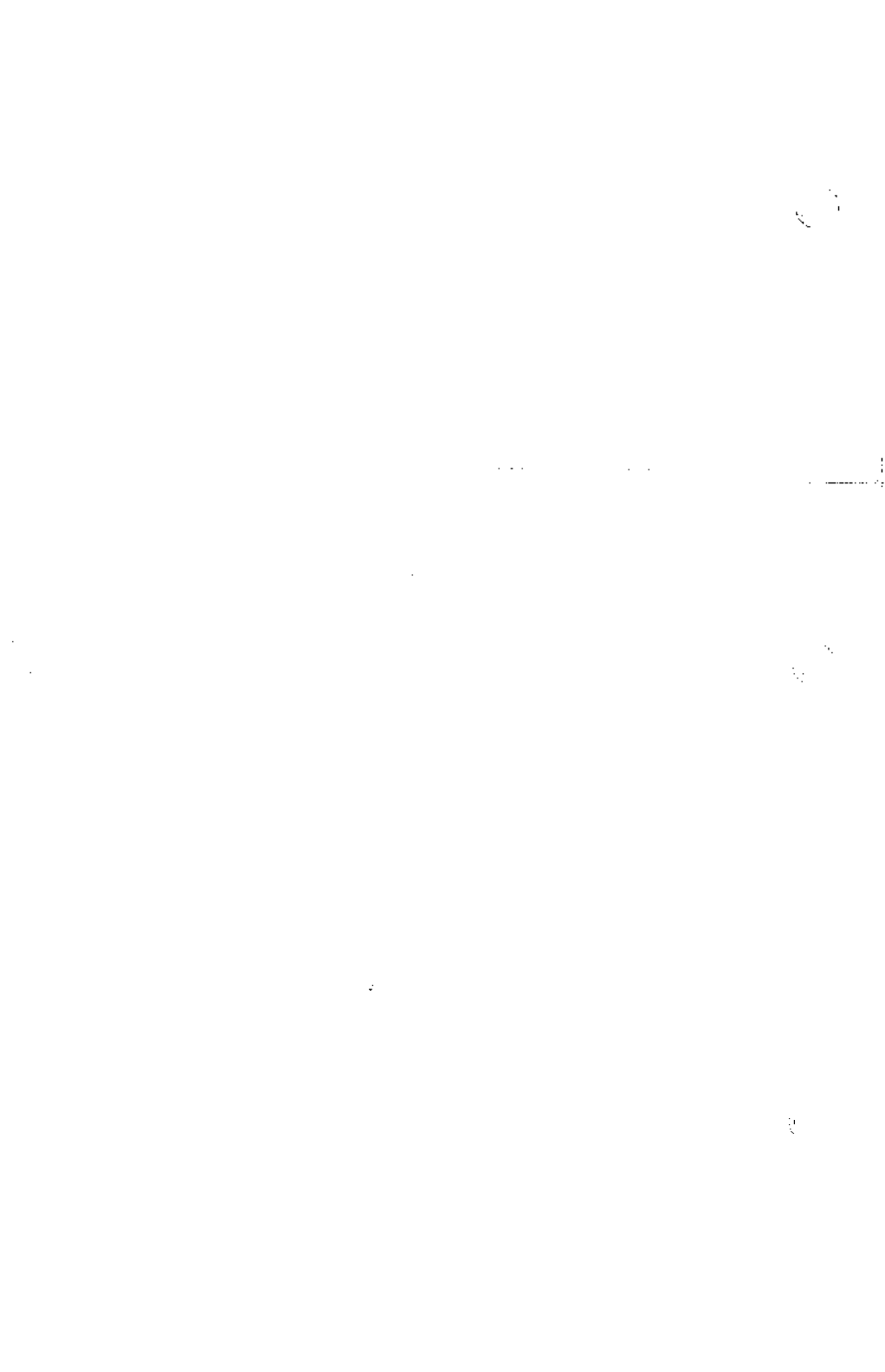
```
view exer4<CR>
<^f>
<^d>
<^b>
<^u>
```

C-7

Chapter 8

ADMINISTRATIVE ADVICE (INTEL PROCESSORS)

	PAGE
GENERAL	1
ADMINISTRATOR'S ROAD MAP	1
CONFIGURATION GUIDELINES	2
DISK FREE SPACE	2
A FEW WORDS ABOUT SYSTEM TUNING	3
PROTECTING USER FILES	4
FILE SYSTEM BACKUP PROGRAMS	4
CONTROLLING DISK USAGE	5
REORGANIZING FILE SYSTEMS	7
KEEPING DIRECTORY FILES SMALL	8
ADMINISTRATIVE USE OF "CRON"	8
WATCH OUT FOR FILES AND DIRECTORIES THAT GROW	9
ALLOCATING RESOURCES TO USERS	10
THE MATTER OF ACCOUNTING AND USAGE	10
DIAL-LINE UTILIZATION	11
"BIRD-DOGGING"	11
TERMINALS	11
LINE PRINTERS	11
SECURITY	12
COMMUNICATING WITH THE USERS	12
TROUBLESHOOTING	13
DATA SET OPTIONS	16
NULL MODEM WIRING	17



ADMINISTRATIVE ADVICE (INTEL PROCESSORS)

GENERAL

The information contained in this chapter is relative to the Intel iAPX 286 processor.

ADMINISTRATOR'S ROAD MAP

This chapter contains administrative advice based on the experience and suggestions of many system administrators. Other reasonable approaches may be taken to solve many of the problem areas described.

Getting started as a UNIX system administrator is hard work. There are no real shortcuts to a working knowledge of the system. The system administrator will need time for reading, studying, and hands-on experimenting. The system administrator should not go "live" with the system until he/she have had several weeks to learn the job and get the initial hardware quirks ironed out.

The administrator should be familiar with a lot of the distributed documentation. The "Introduction" and "UNIX System Capabilities" sections of the this manual should be studied.

Throughout this chapter, each reference of the form **name(N)**, where "N" is the number 1 or 7 possibly followed by a letter, refer to entry **name** in section N of the Runtime System manual. If "N" is a number 2 through 5 possibly followed by a letter, refer to entry **name** in section N of the Software Development System manual.

In these manuals, pay special attention to: **acct(1M)**, **checkall(1M0)**, **chmod(1)**, **chown(1)**, **config(1M)**, **cpio(1)**, **date(1)**, **dcopy(1M)**, **df(1M)**, **don(1M)**, **du(1)**, **ed(1)**, **env(1)**, **errpt(1M)**, **find(1)**, **format(1M)**, **fsock(1M)**, **fuser(1M)**, **kill(1)**, **mail(1)**, **mkdir(1)**, **mkfs(1M)**, **ncheck(1M)**, **ps(1)**, **rm(1)**, **rmdir(1)**, **shutdown(1M)**, **stty(1)**, **su(1)**, **sync(1M)**, **time(1)**, **volcopy(1M)**, **wall(1M)**, **who(1)**, and **write(1)**; **acct(4)**; all of Section 7; and **crash(1)**.

CONFIGURATION GUIDELINES

Minimum configuration requirements for the iAPX 286 are shown in Figure 2-1.

COMPONENT	MINIMUM CAPACITY REQUIRED
Memory:	384 kilobytes, plus 64 kilobytes per terminal.*
Disk space:	10 megabytes, plus 2.5 megabytes per user.*

* **terminal** is the number of users who can be on the system at the same time, and **user** is the number of "average-activity" user accounts.

Figure 2-1. Recommended Configurations

DISK FREE SPACE

Making files is easy under the UNIX operating system. Therefore, users tend to create numerous files using large amounts of file space. It has been said that the only standard thing about all UNIX systems is the message-of-the-day telling users to clean up their files. Administratively, both free disk blocks and free inodes (UNIX system talk for file headers) can be a problem. If the free inode count falls below 100, the system spends most of its time rebuilding the free

inode array. If a file system runs out of space, the system prints "no-space" messages and does little else. To avoid problems, the following start-of-day free counts should be maintained:

- The file system containing */tmp* (temporary files):
 - 1000 free kilobytes.
- The file system containing */usr*:
 - 2000 free kilobytes.
- Other user file systems:
 - 6 to 10 percent free depending on user habits.

This brings up an associated problem: how big should file systems be? The preference is to set aside space on each drive for a copy of root/swap and use the rest of the pack for a single file system. However, if you have user groups that fight over disk space, it may be better to split them up arbitrarily (i.e., divide a pack into more than one file system).

Warning: If different disk drives are set up with differing cylinder partitions between file systems, it will eventually lead to an operational blunder.

A FEW WORDS ABOUT SYSTEM TUNING

A file system reorganization can help throughput but at the expense of down time. If the reorganization is done during nonprime time, it can help.

If normal shutdown and filesave procedures are used, the file system check program [**fsck(1M)**, **-S** option] will help keep the disk free list in reasonable order. Try to keep disk drive usage balanced. If there are over 20 users, the root file system (*/bin*, */tmp*, and */etc*) deserves a drive of its own. If there is a noisy modem (poorly executed do-it-yourself null-modem) or a disconnected modem cable, the UNIX system will spend a lot of CPU time trying to get it logged in. A random check of systems uncovers a lot of this going on.

PROTECTING USER FILES

Users, especially inexperienced ones, occasionally remove their own files. Open files are sometimes lost when the system crashes. Once in a great while, an entire file system will be destroyed (picture a disk controller that goes bad and writes when it should read). Here is a suggested file backup procedure:

- Each day copy all changed user files to backup disks. Keep these disks 3 to 5 days before reusing them.
- Once a week copy each file system to a set of "weekly" disks. Keep weekly disks for 8 weeks.
- Keep bimonthly disks "forever" (they should be recopied once a year).

The most recent weekly disks should be kept off premises. The other disks should be in a fireproof safe if available and not too expensive.

When the UNIX system goes down, active files can get scrambled. Your users will not want to start the day over every time the system fails. In addition to good backup, you *must* have file system patching expertise available (on-site or on-call). If the system is ever rebooted for general use without first checking the file systems, terrible things will happen. Study `checkall(1M)`, `fsck(1M)`, and `crash(1)` as well as the "File System Checking" chapter for more information.

FILE SYSTEM BACKUP PROGRAMS

The following backup programs are distributed:

- **Find/cpio**: The UNIX system is distributed in **cpio** format. The `-cpio` option of the **find** command can be used for saving only those files that have changed or been created over a definite period.
- **Volcopy**: Physical file system copying to disk. For those with a spare drive, **volcopy** to disk provides convenient file restore and quick recovery from disk disasters.

Figure 2-2 summarizes attributes of these programs. In the figure, the file system size is 10,000 KB in all cases; times are in minutes; judgments are subjective.

	FIND/CPIO	VOLCOPY (DISK)	VOLCOPY (FLOPPY)
Full dump time	80	4	60
Incremental dump time	12	-	-
Full restore time	100	4	60
Incremental restore time	15	-	-
Ease of restoring:			
one file	fair	good	fair
a directory	fair	good	fair
scattered files	poor	good	fair
full restore	fair	very good	good
Needs floppy drive	yes	no	yes
Needs spare file system (two CPUs can share)	-	yes	-
Maintains pack/tape labels	no	yes	-
Handles multiple floppies	yes	-	yes
512-byte Blocks per record	10	22	2
Interactive (i.e., ties up console)	yes	yes	yes

Figure 2-2. File System Backup Programs

The spare disk drive is strongly recommended. The speed and convenience of **volcopy** are by no means the only advantage of a spare drive. It is strongly recommended that the administrator modify the */etc/filesave* and */etc/checklist* files to meet the operational needs and update the local operator's manual accordingly. Remember, the more the administrator automates and documents operational procedures the less downtime will be encountered.

CONTROLLING DISK USAGE

If the UNIX system is a success, disk space will soon become limited. During the long delay before more drives become available, usage should be controlled. Try to maintain the start-of-day counts recommended. Watch usage during the day by executing the **df(1)** command regularly.

C-8

ADMINISTRATIVE ADVICE

The **du**(1) command should be executed (after hours) regularly (e.g., daily), and the output kept in an accessible file for later comparison. In this way, users rapidly increasing their disk usage may be spotted. This can also be accomplished by running the accounting system's **acctdusg** program [see **acct(1M)**] as shown in "The UNIX System Accounting" chapter.

The **find**(1) command can be used to locate inactive (or large) files. For example:

```
find / -mtime +90 -atime +90 -print >somefile
```

records in *somefile* the names of files neither written nor accessed in the last 90 days.

The administrator will also have to balance usage between file systems. To do this, user directories must be moved. Users should be taught to accept file system name changes (and to program around them—preferably ahead of time). The user's login directory name (available in the shell variable **HOME**) should be utilized to minimize pathname dependencies. User groups with more extensive file system structures should set up a shell variable to refer to the file system name (e.g., *FS*).

The **find(1)** and **cpio(1)** commands can be used to move user directories and to manipulate the file system tree. The following sequence is useful (it moves the directory trees *userx* and *usery* from file system *filesys1* to file system *filesys2* where, presumably, more space is available):

```
cd /filesys1
find userx usery -print | cpio -pdm /filesys2
# Make sure new copy is OK.
# Change userx and usery login directories
# in the /etc/passwd file.
# Notify userx and usery via mail(1) that
# they have been moved and that pathname
# dependencies in their .profile and shell
# procedures may need changing. See the
# discussion on $HOME above.
rm -rf /filesys1/userx /filesys1/usery
```

When moving more than one user in this way, keep users with common interests in the same file system (these users may have linked files) and move groups of users who may have linked files with a single **cpio** command (otherwise linked files will be unlinked and duplicated).

REORGANIZING FILE SYSTEMS

There is a new file system reorganization utility called **dcopy(1M)**. On an otherwise idle system, a reorganized file system has almost twice the I/O throughput of a randomly organized file system. This applies to file copying, **finds**, **fscks**, etc. **Dcopy** can take up to 2.5 hours to initially reorganize (copy) a large file system. During reorganization, the system can be up, but the file system being copied must be unmounted.

For those who can afford the operator time, root reorganization once a week (requires system reboot) and user file system reorganization once a month will improve system performance. **Dcopy** is an interim step.

KEEPING DIRECTORY FILES SMALL

Directories larger than 5K bytes (320 entries) are very inefficient because of file system indirection. A UNIX system user once complained that it took the system 10 minutes to complete the login process; it turned out that his login directory was 25K bytes long, and the login program spent that time fruitlessly looking for a nonexistent *.profile* file. A large */usr/mail* or */usr/spool/uucp* directory can also really slow the system down. The following will ferret out such directories:

```
find / -type d -size +10 -print
```

Removing files from directories does not make the directories get smaller (the empty directory entries are available for reuse). The following will "compact" */usr/mail* (or any other directory):

```
mv /usr/mail /usr/omail
mkdir /usr/mail
chmod 777 /usr/mail
cd /usr/omail
find . -print | cpio -plm ../mail
cd ..
rm -rf omail
```

ADMINISTRATIVE USE OF "CRON"

The program **cron**(1M) is useful in the administration of the system. It can be used to run the following programs off-hours:

- accounting;
- file system administration;
- long-running, user-written shell procedures.

WATCH OUT FOR FILES AND DIRECTORIES THAT GROW

Most of the files listed below are restarted automatically by entries in */etc/rc* at system reboot.

Accounting Files:

- */etc/wtmp*—login information; grows extremely fast with terminal line difficulties; use **acctcon(1M)** to determine the offending line(s).
- */usr/adm/pacct*—per process accounting records; gets big quickly; monitored automatically by **ckpacct** from **cron(1M)**.
- */usr/lib/cron/log*—status log of commands executed by **cron(1M)**; also watch this file for error messages from the programs being executed in */usr/spool/cron/crontab/**.
- */usr/adm/errfile*—hardware error logging info; also read login **adm's** mail periodically.
- */usr/adm/ctlog*—a log of the people who use **ct(1C)** command.
- */usr/adm/sulog*—a log of those who execute the superuser command.
- */usr/adm/Spacct*—process accounting files left over from an accounting failure; remove these files unless the accounting files that failed are to be rerun.

Other Files:

- */usr/spool*—spooling directory for line printers, **uucp(1C)**, etc., and whose subdirectories should be compacted as described above.

ALLOCATING RESOURCES TO USERS

A prospective user should first obtain authorization to use the system and then apply for a login by providing the following information to the "system administrator":

- User's name.
- Suggested login name (not more than eight characters, beginning with a lowercase letter and not containing special or uppercase letters).
- Relationships to other users (this influences the choice of the file system).
- Estimate of required file space (this also influences the choice of the file system) and connect hours. This aids in hardware growth planning.

Users must have passwords with at least six characters. (Only the first eight characters are significant.) Also, every password must have at least two alphabetic characters and one numeric or special character. The password must differ from the user's login name and any reverse or circular shift of it. Refer to **passwd(1)** and **passwd(4)** for more information on password selection and password aging.

THE MATTER OF ACCOUNTING AND USAGE

You should run the accounting programs even if there is not a "bill" for service. Otherwise, users' habits (especially *bad* habits) will be a mystery to you. Accounting information can also help you find performance bottlenecks, unused logins, bad phone lines, etc.

DIAL-LINE UTILIZATION

If prime-time dial-line utilization gets much over 70 percent, users will start to encounter busy signals when dialing in. This, in turn, will lead to "line hogging". The only solutions are to acquire more dial-up ports, get a larger (another) machine, or get rid of users. Manual policing will help some, but "automatic" policing will be *invariably* subverted by users.

"BIRD-DOGGING"

When the system is busy (lines busy and/or slow response), someone should determine why this is so. The **who(1)** command lists the people logged in. The **ps(1)** command shows what they are doing. Unfortunately, **ps** operates from heuristics that can consistently fail to report certain processes in a busy system. That is, one must be careful about hanging up an apparently inactive line. The **acctcom(1M)** command can read the process accounting file */usr/adm/pacct* backwards from the most recent entry. It will print entries for selected lines or login names.

TERMINALS

Do not use uppercase only terminals. Use full-duplex, full-ASCII asynchronous terminals. Hardware horizontal tabbing is very desirable because it increases output speed and lowers system overhead. A fair proportion of the terminals should provide for correspondence-quality hard copy output to take advantage of the UNIX system word processing capabilities; see **term(5)**.

LINE PRINTERS

Most line printers are troublesome and impose considerable overhead on the system. Most also lack hardware tabs, character overstrike capability, etc. A printer that will work over an asynchronous link (DC1/DC3 protocol required) is the best bet.

SECURITY

The current UNIX operating system is not tamperproof. The system administrator cannot keep people from “breaking” the system but can usually detect that they have done so. The following command will mail (to root) a list of all “set user ID” programs owned by *root* (superuser):

```
find / -user root -perm -4100 -exec ls -l {} \; | mail root
```

Any surprises in *root*'s mail should be investigated. In dealing with security,

- Change the superuser password regularly. Do not pick obvious passwords (choose 6-to-8 character nonsense strings that combine alphabets with digits or special characters).
- Dial ports that do not *require* passwords usually cause trouble.
- The **chroot(1M)** and **su(1)** commands are inherently dangerous as are *group* passwords.
- Login directories, *.profile* files, and files in */bin*, */usr/bin*, */sbin*, and */etc* that are writable by others than their respective owners are security weak spots; police the system regularly against them.
- Remember, no time-sharing system with dial ports is really secure. **Do not keep top secret information on the system.**

COMMUNICATING WITH THE USERS

The directory */usr/news* and the **news(1)** command are provided as a way to get “brief” announcements to your users. More pressing items (one-liners) can be entered in the */etc/motd* (message of the day) file; *motd* and (new to the user) *news* are announced at login time.

To reach users who are already logged in, use the **wall(1M)** (write all) command. Do not use **wall** while logged-in as superuser, except in emergencies.

The */usr/news* directory should be cleaned out once a week by removing everything older than 2 months. It has been found that on most systems a file in */usr/news* will reach 50 percent of the users within a day and over 80 percent within a week; *motd* should be cleaned out daily.

TROUBLESHOOTING

It would be easy to write a book on troubleshooting. The following is some effective advice in dealing with troubles. In dealing with hardware support service personnel,

- Be sure that the contractor agrees to get along with the UNIX software before you take out a hardware service contract ("It's the hardware," says you; "It's the software," says the hardware service contractor).
- Keep on top of problems. Remember that an unreported problem is getting no priority at all. If a problem persists, escalate it through the contractor's local management chain; it may also be effective to complain to the contractor's sales representative.
- Know the details of the support service offering applicable to the installation. In particular, make sure that preventive maintenance is scheduled in advance and that it is completed.
- A "site log" should be maintained for the hardware. All troubles should be recorded in the log by the support service personnel and/or the operating personnel.
- Before changing the approved hardware configuration, make sure that the hardware vendor (as well as the hardware service contractor, if the two are different) agrees to the presence of nonstandard equipment on your system.

ADMINISTRATIVE ADVICE

- Run error logging and maintain console sheets. Make sure error messages are shown to support service personnel.
- Take core dumps after system crashes and have them available for support service personnel.
- Keep records of downtime and make sure that support service personnel know about them.

Telephone problems are most apt to occur when rearranging or adding equipment. Occasionally, central office, trunking, or modem failures occur. In dealing with the telephone services vendor,

- Be specific with repair operators. Tell the operators that the trouble involves *data* equipment.
- If the first call fails to get results, ask for the “supervisor” on the second call, and if necessary, escalate further to get the problem solved.

Some of the obvious problem areas are:

- Disk Drives—Over 50 percent of the problems are likely to be related to the disk subsystem. As mentioned earlier, the way to keep the system *up* is to have a spare disk drive. Remember that preventive maintenance of disk drives is very important. Make sure that the support service personnel who service the hardware see the error-logging printouts and console error messages produced by the UNIX system (and that they understand them). Disk failure can ruin a file system. The *only* defense is to make a complete, daily file backup! (See the part “Protecting User Files”.)
- Dial Ports—In the dial-in interface area, there is room for finger-pointing among all involved vendors. Check for obvious things such as is the system in “multiuser” mode, is the */etc/inittab* file OK, or are any cables loose (*both* ends)? In some telephone offices, trunk hunting is based on 10-number groups. Hunting *between* such groups can fail independently of anything else. The possibilities for trouble are many. Figure 2-3 attempts to describe some alternatives; it is meant

primarily for users of serial communication devices. As an example of the format, (vertical) Rule 3 reads: "If line rings and ring light shows and computer does not answer and switching the modem solves the problem, then it is likely to be a telephone company problem; also, busy out that line."

Rules:		1	2	3	4	5	6	7	8	9	0
Condition:											
	Line rings	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
	Ring light shows on telephone console	-	N	Y	Y	Y	Y	Y	Y	Y	Y
	Computer answers	-	-	N	N	Y	Y	Y	Y	Y	Y
	Login message received on terminal	-	-	-	-	N	N	Y	Y	Y	Y
	Switching modem solves problem	-	-	Y	N	Y	N	-	-	-	-
	User can login	-	-	-	-	-	-	N	N	N	Y
	Telephone console shows data received	-	-	-	-	-	-	Y	Y	N	-
	Problem affects whole serial controller	-	-	-	-	-	-	Y	N	-	-
Diagnosis and/or Action:											
	No problem	-	-	-	-	-	-	-	-	-	X
	Processor hardware problem likely	-	-	-	X	-	X	X	-	-	-
	Telephone problem likely	X	X	X	-	X	-	-	-	X	-
	May be a problem with user's terminal	-	-	-	-	-	-	-	X	-	-
	Busy out bad line(s)	X	X	X	X	X	X	X	-	X	-

Figure 2-3. Asynchronous Line Problems

- Power Supply Modules—There are a lot of them, and they do fail, more or less regularly. Hard failure can be detected at the console; voltage drift is tougher.

C-8

DATA SET OPTIONS

The following data set options seem to work with the UNIX system:

The 801C-L1 (Auto-Call Unit):

Jumpers:

E2 to E3

E6 to E5

Options:

Y, X, T, B,

ZG, ZP, G,

R, ZT

Switches (0 = open, 1 = closed, i.e., side next to number is down):

S1 = 1000[1] (Bracketed switches are missing on some models.)

S2 = 0101

S3 = 11010

S4 = 11[00]

The 212A-L1 (1200-baud full duplex):

Options:

E, ZF, YF, YC,

YG, YJ, YK,

S, V, A, T, ZH,

W, YP, YR

Switches:

S1 = [0]001

S2 = 110001000

S3 = 11110000 (10100000 on 212AR-L1)

S5 = 00

NULL MODEM WIRING

Improperly wired null modems can cause spurious interrupts, especially at higher baud rates. A single bad modem on a 9600-baud line can waste 15 percent of your CPU power. The following (symmetrical) wiring plan will prevent such problems:

pin 1 to 1
pin 2 to 3
pin 3 to 2
strap pin 4 to 5 in the same plug
pin 6 to 20
pin 7 to 7
pin 8 to 20
pin 20 to 6 and 8
ground unused pins

ADMINISTRATIVE FILES

/etc/motd

This file contains the *message-of-the-day*. It is printed by */etc/profile* after every successful *login*; therefore, it should be kept short and to the point.

/etc/brc

This file is executed prior to entering any of the numbered *init* states for the first time after a reboot. The file is generally used to clear the file */etc/mnttab*. It is important to remember this file is executed once per reboot and is controlled by */etc/inittab*.

/etc/powerfail

This shell script is executed according to its line in */etc/inittab*.

/etc/TIMEZONE

This shell script is executed by several initialization procedures. It should be modified to set the correct time zone information for your system.

/etc/bsetdate

This shell script is executed according to its line in */etc/inittab*.

/etc/rc0

This shell script is executed according to its line in */etc/inittab* to run procedures needed to cleanly halt the system.

/etc/rc2

This shell script is executed according to its line in */etc/inittab* to run procedures to bring the system into multiuser mode.

/etc/shutdown.d/*

These shell scripts are executed by */etc/rc0*.

/etc/rc.d/f

These shell scripts are executed by */etc/rc2*.

/etc/inittab

This file is used by */etc/init* to determine the processes to create or terminate in each init state. By convention, state 's' is single user and state '2' is multiuser.

The following line may be used to indicate the default init state, that is, the state the system is to come up in (most likely multiuser).

```
is:2:initdefault:
```

To enable line */dev/tty0* for use by 9600-baud asynchronous terminals, change the following:

```
0:2:off:/etc/getty tty0 9600
```

to

```
0:2:respawn:/etc/getty tty0 9600
```

The arguments to *getty* are the device name optional speed settings which refer to an entry in */etc/gettydefs*, optional type of terminal referenced in *getty(1M)*, and optional line discipline.

To add or delete *getty-login* processes while the system is in multiuser mode, make the appropriate changes to */etc/inittab* then issue the command */etc/init q*. This forces */etc/init* to reread */etc/inittab* without having to change init states.

Again, this file must be edited for local conditions; see **getty(8)**, **init(8)**, **gettydefs(4)**, and **inittab(5)**.

/etc/passwd

This file is used to describe each *user* to the system. A new line must be added for each new user. Each line has seven fields separated by colons:

1. Login name: normally 1 to 8 characters, first character alphabetic, the remainder alphanumeric, no uppercase characters.
2. Encrypted password: initially null, filled in by **passwd(1)**. The encrypted password contains 13 bytes while the actual password is limited to a maximum of 8 bytes. The encrypted password may be followed by a comma and up to four more bytes of password "age" information.
3. User ID: a number between 0 and 65,535; 0 indicates the superuser. User IDs 0 through 99 are reserved.
4. Group ID: the default is group 1 (one). Group IDs 0 through 99 are reserved.
5. Accounting information: this field is used by various accounting programs. It usually contains the user name, department number, and account number.
6. Login directory: full path name (keep them reasonably short).
7. Program name: if null, */bin/sh* is invoked after a successful login. If present, the named program is invoked in place of */bin/sh*.

For example,

```
ghh::138:1:6824-G.H.Hurtz(4357):/usr/ghh:  
grk::244:1:6510-S.P.LeName(4466):/usr/grk:/bin/rsh
```

See also **passwd(4)**, **login(1)**, and **passwd(1)**.

/etc/group

This file is used to describe each *group* to the system. The system administrator must add a new line for each new group. Each line has four fields separated by colons:

1. Group name: normally 1 to 8 characters, first character alphabetic, the remainder alphanumeric, no uppercase characters.
2. Encrypted password: contains 13 bytes while the actual password is limited to a maximum of 8 bytes.
3. Group ID: a number between 0 and 65,535. Group IDs 0 through 99 are reserved.
4. Login names: list of all *login* names in the group, separated by commas; list of all *login* names that may use **newgrp(1)** to become a member of the group.

Group passwords are strongly discouraged. See also **group(4)**.

/etc/profile

When the shell is executed and is the leader of a process group, as is the case when it is invoked by *login*, it will read and execute the commands in */etc/profile* before executing commands in the user's *.profile* file. This allows the system administrator to set up a standard environment for all users (e.g., executing **umask**, setting shell variables, etc.) and take care of other housekeeping details (such as **news -n**). Note that in */etc/profile* the shell variable **\$0** indicates the invocation—normal shell (-sh), restricted shell (-rsh), or **su** command (-su).

/etc/checklist

This file contains a list of default devices to be checked for consistency by the **fsck(1M)** program. The devices normally correspond to those mounted when the system is in *multiuser* mode. For example, a sample checklist would be:

```
/dev/dsk/0s6  
/dev/rdisk/0s7
```

Note that the *root* device is specified as a *block* device while all others are specified as *character* devices. Character devices can be checked faster than block devices. The *root* device is specified as a block device in order for the **fsck** program to detect when the *root* is being checked, so that any modifications to this file system will result in an immediate reboot request.

/etc/fstab

This file contains a list of devices and mount points that are used by */etc/rc.d/MOUNT.rc* to check and mount all of the file systems except *root*. This file must be modified whenever the set of file systems to be mounted at boot time changes.

/etc/shutdown

This file contains procedures to gracefully shut down the system in preparation for file save or scheduled downtime. Beware that no procedures appear after the transition to single-user mode, as it may not be completed before the transition occurs.

/etc/filesave

This file contains prototypes for local file saves.

/usr/adm/pacct

This file contains the process accounting information; see **acct(1M)**.

/etc/wtmp

This file is the log of *login* processes.



Chapter 9

FILE SYSTEM CHECKING

	PAGE
GENERAL	1
System Administrator Advice	2
UPDATE OF THE FILE SYSTEM	2
Superblock	2
Inodes	3
Indirect Blocks	3
Data Blocks	3
First Free-List Block	3
CORRUPTION OF THE FILE SYSTEM	4
Improper System Shutdown and Startup	4
Hardware Failure	4
DETECTION AND CORRECTION OF CORRUPTION	4
Superblock	5
Inodes	6
Indirect Blocks	9
Data Blocks	10
Free-List Blocks	11
FSCCK Error Conditions	13



FILE SYSTEM CHECKING

The File System Check Program (**fsck**) is an interactive file system check and repair program. **Fsck** uses the redundant structural information in the UNIX system file system to perform several consistency checks. If an inconsistency is detected, it is reported to the operator, who may elect to fix or ignore each inconsistency. These inconsistencies result from the permanent interruption of the file system updates, which are performed every time a file is modified. **Fsck** is frequently able to repair corrupted file systems using procedures based upon the order in which the UNIX system honors these file system update requests.

The purpose of this chapter is to describe the normal updating of the file system, to discuss the possible causes of file system corruption, and to present the corrective actions implemented by **fsck**. Both the program and the interaction between the program and the operator are described.

Appendix 6-1 contains the **fsck** error conditions. The meanings of the various error conditions, possible responses, and related error conditions are explained.

C-9

GENERAL

When a UNIX operating system is brought up, a consistency check of the file systems should always be performed. This precautionary measure helps to ensure a reliable environment for file storage on disk. If an inconsistency is discovered, corrective action must be taken.

The updating of the file system and file system corruption is described in this chapter. Finally, the set of heuristically sound corrective actions used by **fsck** are presented.

System Administrator Advice

Remember that system buffers are 1024 bytes. When configuring the operating system, take into consideration that the same number of buffers as before will use more main memory. Weigh this against reducing the number of buffers, which reduces the cache hit ratio and degrades performance.

UPDATE OF THE FILE SYSTEM

Every working day hundreds of files are created, modified, and removed. Every time a file is modified, the UNIX operating system performs a series of file system updates. These updates, when written on disk, yield a consistent file system. To understand what happens in the event of a permanent interruption in this sequence, it is important to understand the order in which the update requests were probably being honored. Knowing which pieces of information were probably written to the file system first, heuristic procedures can be developed to repair a corrupted file system.

There are five types of file system updates. These involve the superblock, inodes, indirect blocks, data blocks (directories and files), and free-list blocks.

Superblock

The superblock contains information about the size of the file system, the size of the inode list, part of the free-block list, the count of free blocks, the count of free inodes, and part of the free-inode list.

The superblock of a mounted file system (the root file system is always mounted) is written to the file system whenever the file system is unmounted or a **sync** command is issued.

Inodes

An inode contains information about the type of inode (directory, data, or special), the number of directory entries linked to the inode, the list of blocks claimed by the inode, and the size of the inode.

An inode is written to the file system upon closure of the file associated with the inode. (All "in" core blocks are also written to the file system upon issue of a **sync** system call.)

Indirect Blocks

There are three types of indirect blocks—single-indirect, double-indirect, and triple-indirect. A single-indirect block contains a list of some of the block numbers claimed by an inode. Each one of the 128 entries in an indirect block is a data-block number. A double-indirect block contains a list of single-indirect block numbers. A triple-indirect block contains a list of double-indirect block numbers.

Indirect blocks are written to the file system whenever they have been modified and released by the operating system. More precisely, they are queued for eventual writing. Physical I/O is deferred until the buffer is needed by the UNIX system or a **sync** command is issued.

Data Blocks

A data block may contain file information or directory entries. Each directory entry consists of a file name and an inode number.

Data blocks are written to the file system whenever they have been modified and released by the operating system.

First Free-List Block

The superblock contains the first free-list block. The free-list blocks are a list of all blocks that are not allocated to the superblock, inodes, indirect blocks, or data blocks. Each free-list block contains a count of the number of entries in this free-list block, a pointer to the next free-list block, and a partial list of free blocks in the file system.

Free-list blocks are written to the file system whenever they have been modified and released by the operating system.

CORRUPTION OF THE FILE SYSTEM

A file system can become corrupted in a variety of ways. Improper shutdown procedures and hardware failures are the most common.

Improper System Shutdown and Startup

File systems may become corrupted when proper shutdown procedures are not observed, e.g., forgetting to **sync** the system prior to halting the CPU, physically write-protecting a mounted file system, or taking a mounted file system off-line.

File systems may also become further corrupted by allowing a corrupted file system to be used (and, thus, to be modified further) can be disastrous.

Hardware Failure

Any piece of hardware can fail at any time. Failures can be as subtle as a bad block on a disk platter or as blatant as a nonfunctional disk controller.

DETECTION AND CORRECTION OF CORRUPTION

A quiescent file system (an unmounted system and not being written on) may be checked for structural integrity by performing consistency checks on the redundant data intrinsic to a file system. The redundant data is either read from the file system or computed from other known values. A quiescent state is important during the checking of a file system because of the multipass nature of the **fsck** program.

When an inconsistency is discovered, **fsck** reports the inconsistency for the operator to choose a corrective action.

Discussed in this part are how to discover inconsistencies (and possible corrective actions) for the superblock, the inodes, the indirect blocks, the data blocks containing directory entries, and the free-list blocks. These corrective actions can be performed interactively by the **fsck** command under control of the operator.

Superblock

One of the most common corrupted items is the superblock. The superblock is prone to corruption because every change to the file system's blocks or inodes modifies the superblock.

The superblock and its associated parts are most often corrupted when the computer is halted and the last command involving output to the file system was not a **sync** command.

The superblock can be checked for inconsistencies involving file system size, inode-list size, free-block list, free-block count, and the free-inode count.

File System Size and Inode-List Size

The file system size must be larger than the number of blocks used by the superblock and the number of blocks used by the list of inodes. The number of inodes must be less than 65,535. The file system size and inode-list size are critical pieces of information to the **fsck** program. While there is no way to actually check these sizes, **fsck** can check for them being within reasonable bounds. All other checks of the file system depend on the correctness of these sizes.

Free-Block List

The free-block list starts in the superblock and continues through the free-list blocks of the file system. Each free-list block can be checked for a list count out of range, for block numbers out of range, and for blocks already allocated within the file system. A check is made to see that all the blocks in the file system were found.

The first free-block list is in the superblock. **Fsck** checks the list count for a value of less than 0 or greater than 50. It also checks each block number for a value of less than the first data block in the file system or greater than the last block in the file system. Then it compares each block number to a list of already allocated blocks. If the free-list block pointer is nonzero, the next free-list block is read in and the process is repeated.

When all the blocks have been accounted for, a check is made to see if the number of blocks used by the free-block list plus the number of blocks claimed by the inodes equals the total number of blocks in the file system.

If anything is wrong with the free-block list, then **fsck** may rebuild the list, excluding all blocks in the list of allocated blocks.

Free-Block Count

The superblock contains a count of the total number of free blocks within the file system. **Fsck** compares this count to the number of blocks it found free within the file system. If the counts do not agree, then **fsck** may replace the count in the superblock by the actual free-block count.

Free-Inode Count

The superblock contains a count of the total number of free inodes within the file system. **Fsck** compares this count to the number of inodes it found free within the file system. If the counts do not agree, then **fsck** may replace the count in the superblock by the actual free-inode count.

Inodes

An individual inode is not as likely to be corrupted as the superblock. However, because of the great number of active inodes, there is almost as likely a chance for corruption in the inode list as in the superblock.

The list of inodes is checked sequentially starting with inode 1 (there is no inode 0) and going to the last inode in the file system. Each inode can be checked for inconsistencies involving format and type, link count, duplicate blocks, bad blocks, and inode size.

Format and Type

Each inode contains a mode word. This mode word describes the type and state of the inode. Inodes may be one of four types:

- Regular
- Directory
- Special block
- Special character.

If an inode is not one of these types, then the inode has an illegal type. Inodes may be found in one of three states—unallocated, allocated, and neither unallocated nor allocated. This last state indicates an incorrectly formatted inode. An inode can get in this state if bad data is written into the inode list through, for example, a hardware failure. The only possible corrective action is for **fsck** to clear the inode.

Link Count

Contained in each inode is a count of the total number of directory entries linked to the inode. **Fsck** verifies the link count of each inode by traversing down the total directory structure, starting from the root directory, and calculating an actual link count for each inode.

If the stored link count is nonzero and the actual link count is zero, it means that no directory entry appears for the inode. If the stored and actual link counts are nonzero and unequal, a directory entry may have been added or removed without the inode being updated.

If the stored link count is nonzero and the actual link count is zero, **fsck** can, under operator control, link the disconnected file to the *lost+found* directory. If the stored and actual link counts are nonzero and unequal, **fsck** can replace the stored link count by the actual link count.

Duplicate Blocks

Contained in each inode is a list or pointers to lists (indirect blocks) of all the blocks claimed by the inode. **Fsck** compares each block number claimed by an inode to a list of already allocated blocks. If a block number is already claimed by another inode, the block number is added to a list of duplicate blocks. Otherwise, the list of allocated blocks is updated to include the block number. If there are any duplicate blocks, **fsck** will make a partial second pass of the inode list to find the inode of the duplicated block. This is necessary because without examining the files associated with these inodes for correct content there is not enough information available to decide which inode is corrupted and should be cleared. Most of the time, the inode with the earliest modify time is incorrect and should be cleared. This condition can occur by using a file system with blocks claimed by both the free-block list and by other parts of the file system.

A large number of duplicate blocks in an inode may be due to an indirect block not being written to the file system. **Fsck** will prompt the operator to clear both inodes.

Bad Blocks

Contained in each inode is a list or pointer to lists of all the blocks claimed by the inode. **Fsck** checks each block number claimed by an inode for a value lower than that of the first data block or greater than the last block in the file system. If the block number is outside this range, the block number is a bad block number.

If there is a large number of bad blocks in an inode, this may be due to an indirect block not being written to the file system. **Fsck** will prompt the operator to clear both inodes.

Size Checks

Each inode contains a 32-bit (4-byte) size field. This size indicates the number of characters in the file associated with the inode. This size can be checked for inconsistencies, e.g., directory sizes that are not a multiple of 16 characters or the number of blocks actually used not matching that indicated by the inode size.

A directory inode within the file system has the directory bit on in the inode mode word. The directory size must be a multiple of 16 because a directory entry contains 16 bytes (2 bytes for the inode number and 14 bytes for the file or directory name).

Fsck will warn of such directory misalignment. This is only a warning because not enough information can be gathered to correct the misalignment.

A rough check of the consistency of the size field of an inode can be performed by computing from the size field the number of blocks that should be associated with the inode and comparing it to the actual number of blocks claimed by the inode.

Fsck calculates the number of blocks that there should be in an inode by dividing the number of characters in an inode by the number of characters per block and rounding up. **Fsck** adds one block for each indirect block associated with the inode. If the actual number of blocks does not match the computed number of blocks, **fsck** will warn of a possible file-size error. This is only a warning because the UNIX system does not fill in blocks in files created in random order.

Indirect Blocks

Indirect blocks are owned by an inode. Therefore, inconsistencies in indirect blocks directly affect the inode that owns it.

Inconsistencies that can be checked are blocks already claimed by another inode and block numbers outside the range of the file system.

For a discussion of detection and correction of the inconsistencies associated with indirect blocks, see parts "Duplicate Blocks" and "Bad Blocks".

Data Blocks

The two types of data blocks are plain data blocks and directory data blocks. Plain data blocks contain the information stored in a file. Directory data blocks contain directory entries. **Fsck** does not attempt to check the validity of the contents of a plain data block.

Each directory data block can be checked for inconsistencies involving directory inode numbers pointing to unallocated inodes, directory inode numbers greater than the number of inodes in the file system, incorrect directory inode numbers for "." and "..", and directories disconnected from the file system. In addition, the validity of the contents of a directory's data block is checked.

If a directory entry inode number points to an unallocated inode, then **fsck** may remove that directory entry. This condition probably occurred because the data blocks containing the directory entries were modified and written out while the inode was not yet written out.

If a directory entry inode number is pointing beyond the end of the inode list, **fsck** may remove that directory entry. This condition occurs if bad data is written into a directory data block.

The directory inode number entry for "." should be the first entry in the directory data block. Its value should be equal to the inode number for the directory data block.

The directory inode number entry for ".." should be the second entry in the directory data block. Its value should be equal to the inode number for the parent of the directory entry (or the inode number of the directory data block if the directory is the root directory).

If the directory inode numbers are incorrect, **fsck** may replace them with the correct values.

Fck checks the general connectivity of the file system. If directories are found not to be linked into the file system, **fck** will link the directory back into the file system in the *lost+found* directory. This condition can be caused by inodes being written to the file system with the corresponding directory data blocks not being written to the file system.

Free-List Blocks

Free-list blocks are owned by the superblock. Therefore, inconsistencies in free-list blocks directly affect the superblock.

Inconsistencies that can be checked are a list count outside of range, block numbers outside of range, and blocks already associated with the file system.

For a discussion of detection and correction of the inconsistencies associated with free-list blocks, see part "Free-Block List".

(BLANK)

FSCK ERROR CONDITIONS

A. Conventions

Fsck is a multipass file system check program. Each file system pass invokes a different phase of the **fsck** program. After the initial setup, **fsck** performs successive phases over each file system performing cleanup, checking blocks and sizes, path names, connectivity, reference counts, and the free-block list (possibly rebuilding it).

When an inconsistency is detected, **fsck** reports the error condition to the operator. If a response is required, **fsck** prints a prompt message and waits for a response. This appendix explains the meaning of each error condition, the possible responses, and the related error conditions.

The error conditions are organized by the "Phase" of the **fsck** program in which they can occur. The error conditions that may occur in more than one phase will be discussed under Part B.

B. Initialization

Before a file system check can be performed, certain tables have to be set up and certain files opened. This section describes the opening of files and the initialization of tables. Error conditions resulting from command line options, memory requests, opening of files, status of files, file system size checks, and creation of the scratch file are listed below.

C option?

C is not a legal option to **fsck**; legal options are **-y**, **-n**, **-s**, **-S**, **-t**, **-b**, **-r**, **-q**, and **-D**. **Fsck** terminates on this error condition. See the **fsck(1M)** entry in the *Runtime System manual* for further details.

Bad -t option

The **-t** option is not followed by a file name. **Fsck** terminates on this error condition. See the **fsck(1M)** entry in the *Runtime System manual* for further details.

Invalid `-s` argument, defaults assumed

The `-s` option is not suffixed by 3, 4, or blocks-per-cylinder: blocks-to-skip. **Fsck** assumes a default value of 400 blocks-per-cylinder and 9 blocks-to-skip. See the **fsck(1M)** entry in the *Runtime System manual* for further details.

Incompatible options: `-n` and `-s`

It is not possible to salvage the free-block list without modifying the file system. **Fsck** terminates on this error condition. See the **fsck(1M)** entry in the *Runtime System manual* for further details.

Can not `fstat` standard input

Fsck's attempt to `fstat` standard input failed. The occurrence of this error condition indicates a serious problem which may require additional assistance. **Fsck** terminates on this error condition.

Can not get memory

Fsck's request for memory for its virtual memory tables failed. The occurrence of this error condition indicates a serious problem which may require additional assistance. **Fsck** terminates on this error condition.

Can not open checkall file: `F`

The default file system checkall file *F* (usually */etc/checkall*) cannot be opened for reading. **Fsck** terminates on this error condition. Check access modes of *F*.

Can not `stat` root

Fsck's request for statistics about the root directory `"/` failed. The occurrence of this error condition indicates a serious problem which may require additional assistance. **Fsck** terminates on this error condition.

Can not `stat F`

Fsck's request for statistics about the file system *F* failed. It ignores this file system and continues checking the next file system given. Check access modes of *F*.

F is not a block or character device

Fsck has been given a regular file name by mistake. It ignores this file system and continues checking the next file system given. Check file type of *F*.

Can not open F

The file system *F* cannot be opened for reading. It ignores this file system and continues checking the next file system given. Check access modes of *F*.

Size check: fsize X isize Y

More blocks are used for the inode list *Y* than there are blocks in the file system *X*, or there are more than 65,535 inodes in the file system. It ignores this file system and continues checking the next file system given.

Can not create F

Fsck's request to create a scratch file *F* failed. It ignores this file system and continues checking the next file system given. Check access modes of *F*.

CAN NOT SEEK: BLK B (CONTINUE)

Fsck's request for moving to a specified block number *B* in the file system failed. The occurrence of this error condition indicates a serious problem which may require additional assistance.

Possible responses to CONTINUE prompt are:

- | | |
|-----|---|
| YES | Attempt to continue to run file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of fsck should be made to recheck this file system. If block was part of the virtual memory buffer cache, fsck will terminate with the message "Fatal I/O error". |
| NO | Terminate program. |

FCK

CAN NOT READ: BLK B (CONTINUE)

Fck's request for reading a specified block number *B* in the file system failed. The occurrence of this error condition indicates a serious problem which may require additional assistance.

Possible responses to CONTINUE prompt are:

YES Attempt to continue to run file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system. If block was part of the virtual memory buffer cache, **fsck** will terminate with the message "Fatal I/O error".

NO Terminate program.

CAN NOT WRITE: BLK B (CONTINUE)

Fck's request for writing a specified block number *B* in the file system failed. The disk is write-protected.

Possible responses to CONTINUE prompt are:

YES Attempt to continue to run file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system. If block was part of the virtual memory buffer cache, **fsck** will terminate with the message "Fatal I/O error".

NO Terminate program.

C. PHASE 1: CHECK BLOCKS AND SIZES

This phase concerns itself with the inode list. This part lists error conditions resulting from checking inode types, setting up the zero-link-count table, examining inode block numbers for bad or duplicate blocks, checking inode size, and checking inode format.

UNKNOWN FILE TYPE I=I (CLEAR)

The mode word of the inode *I* indicates that the inode is not a special character inode, regular inode, or directory inode.

Possible responses to CLEAR prompt are:

- | | |
|-----|--|
| YES | Deallocate inode <i>I</i> by zeroing its contents. This will always invoke the UNALLOCATED error condition in Phase 2 for each directory entry pointing to this inode. |
| NO | Ignore this error condition. |

LINK COUNT TABLE OVERFLOW (CONTINUE)

An internal table for **fsck** containing allocated inodes with a link count of zero has no more room. Recompile **fsck** with a larger value of MAXLNCNT.

Possible responses to CONTINUE prompt are:

- | | |
|-----|---|
| YES | Continue with program. This error condition will not allow a complete check of the file system. A second run of fsck should be made to recheck this file system. If another allocated inode with a zero link count is found, this error condition is repeated. |
| NO | Terminate program. |

B BAD I=I

Inode *I* contains block number *B* with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system. This error condition may invoke the EXCESSIVE BAD BLKS error condition in Phase 1 if inode *I* has too many block numbers outside the file system range. This error condition will always invoke the BAD/DUP error condition in Phase 2 and Phase 4.

EXCESSIVE BAD BLKS I=I (CONTINUE)

There is more than a tolerable number (usually 10) of blocks with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system associated with inode *I*.

Possible responses to CONTINUE prompt are:

YES Ignore the rest of the blocks in this inode and continue checking with next inode in the file system. This error condition will not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system.

NO Terminate program.

B DUP I=I

Inode *I* contains block number *B* which is already claimed by another inode. This error condition may invoke the EXCESSIVE DUP BLKS error condition in Phase 1 if inode *I* has too many block numbers claimed by other inodes. This error condition will always invoke Phase 1b and the BAD/DUP error condition in Phase 2 and Phase 4.

EXCESSIVE DUP BLKS I=I (CONTINUE)

There is more than a tolerable number (usually 10) of blocks claimed by other inodes.

Possible responses to CONTINUE prompt are:

YES Ignore the rest of the blocks in this inode and continue checking with next inode in the file

system. This error condition will not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system.

NO Terminate program.

DUP TABLE OVERFLOW (CONTINUE)

An internal table in **fsck** containing duplicate block numbers has no more room. Recompile **fsck** with a larger value of DUPTBL5IZE.

Possible responses to CONTINUE prompt are:

YES Continue with program. This error condition will not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system. If another duplicate block is found, this error condition will repeat.

NO Terminate program.

POSSIBLE FILE SIZE ERROR I=I

The inode *I* size does not match the actual number of blocks used by the inode. This is only a warning. If the **-q** option is used, this message is not printed.

DIRECTORY MISALIGNED I=I

The size of a directory inode is not a multiple of the size of a directory entry (usually 16). This is only a warning. If the **-q** option is used, this message is not printed.

PARTIALLY ALLOCATED INODE I=I (CLEAR)

Inode *I* is neither allocated nor unallocated.

Possible responses to CLEAR prompt are:

YES Deallocate inode **I** by zeroing its contents.

NO Ignore this error condition.

D. PHASE 1B: RESCAN FOR MORE DUPS

When a duplicate block is found in the file system, the file system is rescanned to find the inode which previously claimed that block. This part lists the error condition when the duplicate block is found.

B DUP I=I

Inode *I* contains block number *B* which is already claimed by another inode. This error condition will always invoke the BAD/DUP error condition in Phase 2. Inodes with overlapping blocks may be determined by examining this error condition and the DUP error condition in Phase 1.

E. PHASE 2: CHECK PATHNAMES

This phase concerns itself with removing directory entries pointing to error conditioned inodes from Phase 1 and Phase 1b. This part lists error conditions resulting from root inode mode and status, directory inode pointers in range, and directory entries pointing to bad inodes.

ROOT INODE UNALLOCATED. TERMINATING

The root inode (always inode number 2) has no allocate mode bits. The occurrence of this error condition indicates a serious problem which may require additional assistance. The program will terminate.

ROOT INODE NOT DIRECTORY (FIX)

The root inode (usually inode number 2) is not directory inode type.

Possible responses to **FIX** prompt are:

YES Replace the root inode's type to be a directory. If the root inode's data blocks are not directory blocks, a *very* large number of error conditions will be produced.

NO Terminate program.

DUPS/BAD IN ROOT INODE (CONTINUE)

Phase 1 or Phase 1b have found duplicate blocks or bad blocks in the root inode (usually inode number 2) for the file system.

Possible responses to CONTINUE prompt are:

YES Ignore DUPS/BAD error condition in root inode and attempt to continue to run the file system check. If root inode is not correct, then this may result in a large number of other error conditions.

NO Terminate program.

I OUT OF RANGE I=I NAME=F (REMOVE)

A directory entry *F* has an inode number *I* which is greater than the end of the inode list.

Possible responses to REMOVE prompt are:

YES The directory entry *F* is removed.

NO Ignore this error condition.

UNALLOCATED I=I OWNER=O MODE=M SIZE=S MTIME=T NAME=F (REMOVE)

A directory entry *F* has an inode *I* without allocate mode bits. The owner *O*, mode *M*, size *S*, modify time *T*, and file name *F* are printed. If the file system is not mounted and the **-n** option was not specified, the entry will be removed automatically if the inode it points to is character size 0.

Possible responses to REMOVE prompt are:

YES The directory entry *F* is removed.

NO Ignore this error condition.

**DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T
DIR=F (REMOVE)**

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry *F*, directory inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to REMOVE prompt are:

YES The directory entry *F* is removed.

NO Ignore this error condition.

**DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T
FILE=F (REMOVE)**

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry *F*, inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and file name *F* are printed.

Possible responses to REMOVE prompt are:

YES The directory entry *F* is removed.

NO Ignore this error condition.

**BAD BLK B IN DIR I=I OWNER=O MODE=M SIZE=S
MTIME=T**

This message only occurs when the `-q` option is used. A bad block was found in DIR inode *I*. Error conditions looked for in directory blocks are nonzero padded entries, inconsistent "." and ".." entries, and imbedded slashes in the name field. This error message indicates that the user should at a later time either remove the directory inode if the entire block looks bad or change (or remove) those directory entries that look bad.

F. PHASE 3: CHECK CONNECTIVITY

This phase concerns itself with the directory connectivity seen in Phase 2. This part lists error conditions resulting from unreferenced directories and missing or full *lost+found* directories.

**UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T
(RECONNECT)**

The directory inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. **Fsck** will force the reconnection of a nonempty directory.

Possible responses to RECONNECT prompt are:

YES Reconnect directory inode *I* to the file system in directory for lost files (usually *lost+found*). This may invoke *lost+found* error condition in Phase 3 if there are problems connecting directory inode *I* to *lost+found*. This may also invoke CONNECTED error condition in Phase 3 if link was successful.

NO Ignore this error condition. This will always invoke UNREF error condition in Phase 4.

SORRY. NO lost+found DIRECTORY

There is no *lost+found* directory in the root directory of the file system; **fsck** ignores the request to link a directory in *lost+found*. This will always invoke the UNREF error condition in Phase 4. Check access modes of *lost+found*. See **fsck(1M)** in the *UNIX System V Administrator Reference Manual* for further details.

SORRY. NO SPACE IN lost+found DIRECTORY

There is no space to add another entry to the *lost+found* directory in the root directory of the file system; **fsck** ignores the request to link a directory in *lost+found*. This will always invoke the UNREF error condition in Phase 4. Clean out unnecessary entries in *lost+found* or make *lost+found* larger. See **fsck(1M)** in the *UNIX System V Administrator Reference Manual* for further details.

DIR I=I1 CONNECTED. PARENT WAS I=I2

This is an advisory message indicating a directory inode *I1* was successfully connected to the *lost+found* directory. The parent inode *I2* of the directory inode *I1* is replaced by the inode number of the *lost+found* directory.

G. PHASE 4: CHECK REFERENCE COUNTS

This phase concerns itself with the link count information seen in Phase 2 and Phase 3. This part lists error conditions resulting from unreferenced files; missing or full *lost+found* directory; incorrect link counts for files, directories, or special files; unreferenced files and directories; bad and duplicate blocks in files and directories; and incorrect total free-inode counts.

UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)

Inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. If the `-n` option is not set and the file system is not mounted, empty files will not be reconnected and will be cleared automatically.

Possible responses to RECONNECT prompt are:

YES Reconnect inode *I* to file system in the directory for lost files (usually *lost+found*). This may invoke *lost+found* error condition in Phase 4 if there are problems connecting inode *I* to *lost+found*.

NO Ignore this error condition. This will always invoke CLEAR error condition in Phase 4.

SORRY. NO *lost+found* DIRECTORY

There is no *lost+found* directory in the root directory of the file system; `fsck` ignores the request to link a file in *lost+found*. This will always invoke CLEAR error condition in Phase 4. Check access modes of *lost+found*.

SORRY. NO SPACE IN *lost+found* DIRECTORY

There is no space to add another entry to the *lost+found* directory in the root directory of the file system; `fsck` ignores the request to link a file in *lost+found*. This will always invoke the CLEAR error condition in Phase 4. Check size and contents of *lost+found*.

(CLEAR)

The inode mentioned in the immediately previous error condition cannot be reconnected.

Possible responses to CLEAR prompt are:

- YES Deallocate inode mentioned in the immediately previous error condition by zeroing its contents.
- NO Ignore this error condition.

**LINK COUNT FILE I=I OWNER=O MODE=M SIZE=S
MTIME=T COUNT=X SHOULD BE Y (ADJUST)**

The link count for inode *I*, which is a file, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* are printed.

Possible responses to ADJUST prompt are:

- YES Replace link count of file inode *I* with *Y*.
- NO Ignore this error condition.

**LINK COUNT DIR I=I OWNER=O MODE=M SIZE=S
MTIME=T COUNT=X SHOULD BE Y (ADJUST)**

The link count for inode *I*, which is a directory, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed.

Possible responses to ADJUST prompt are:

- YES Replace link count of directory inode *I* with *Y*.
- NO Ignore this error condition.

**LINK COUNT F I=I OWNER=O MODE=M SIZE=S
MTIME=T COUNT=X SHOULD BE Y (ADJUST)**

The link count for *F* inode *I* is *X* but should be *Y*. The file name *F*, owner *O*, mode *M*, size *S*, and modify time *T* are printed.

Possible responses to ADJUST prompt are:

YES Replace link count of inode *I* with *Y*.

NO Ignore this error condition.

**UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T
(CLEAR)**

Inode *I*, which is a file, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. If the `-n` option is not set and the file system is not mounted, empty files will be cleared automatically.

Possible responses to CLEAR prompt are:

YES Deallocate inode *I* by zeroing its contents.

NO Ignore this error condition.

**UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T
(CLEAR)**

Inode *I*, which is a directory, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. If the `-n` option is not set and the file system is not mounted, empty directories will be cleared automatically. Nonempty directories will not be cleared.

Possible responses to CLEAR prompt are:

YES Deallocate inode *I* by zeroing its contents.

NO Ignore this error condition.

**BAD/DUP FILE I=I OWNER=O MODE=M SIZE=S
MTIME=T (CLEAR)**

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with file inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed.

Possible responses to CLEAR prompt are:

YES Deallocate inode *I* by zeroing its contents.

NO Ignore this error condition.

**BAD/DUP DIR I=I OWNER=O MODE=M SIZE=S
MTIME=T (CLEAR)**

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed.

Possible responses to CLEAR prompt are:

YES Deallocate inode *I* by zeroing its contents.

NO Ignore this error condition.

FREE INODE COUNT WRONG IN SUPERBLK (FIX)

The actual count of the free inodes does not match the count in the superblock of the file system. If the **-q** option is specified, the count will be fixed automatically in the superblock.

Possible responses to FIX prompt are:

YES Replace count in superblock by actual count.

NO Ignore this error condition.

H. PHASE 5: CHECK FREE LIST

This phase concerns itself with the free-block list. This part lists error conditions resulting from bad blocks in the free-block list, bad free-blocks count, duplicate blocks in the free-block list, unused blocks from the file system not in the free-block list, and the total free-block count incorrect.

EXCESSIVE BAD BLKS IN FREE LIST (CONTINUE)

The free-block list contains more than a tolerable number (usually 10) of blocks with a value less than the first data block in the file system or greater than the last block in the file system.

Possible responses to CONTINUE prompt are:

YES Ignore rest of the free-block list and continue execution of **fsck**. This error condition will always invoke "BAD BLKS IN FREE LIST" error condition in Phase 5.

NO Terminate program.

EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE)

The free-block list contains more than a tolerable number (usually 10) of blocks claimed by inodes or earlier parts of the free-block list.

Possible responses to CONTINUE prompt are:

YES Ignore the rest of the free-block list and continue execution of **fsck**. This error condition will always invoke "DUP BLKS IN FREE LIST" error condition in Phase 5.

NO Terminate program.

BAD FREEBLK COUNT

The count of free blocks in a free-list block is greater than 50 or less than 0. This error condition will always invoke the "BAD FREE LIST" condition in Phase 5.

X BAD BLKS IN FREE LIST

X blocks in the free-block list have a block number lower than the first data block in the file system or greater than the last block in the file system. This error condition will always invoke the "BAD FREE LIST" condition in Phase 5.

X DUP BLKS IN FREE LIST

X blocks claimed by inodes or earlier parts of the free-list block were found in the free-block list. This error condition will always invoke the "BAD FREE LIST" condition in Phase 5.

X BLK(S) MISSING

X blocks unused by the file system were not found in the free-block list. This error condition will always invoke the "BAD FREE LIST" condition in Phase 5.

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX)

The actual count of free blocks does not match the count in the superblock of the file system.

Possible responses to FIX prompt are:

- | | |
|-----|--|
| YES | Replace count in superblock by actual count. |
| NO | Ignore this error condition. |

BAD FREE LIST (SALVAGE)

Phase 5 has found bad blocks in the free-block list, duplicate blocks in the free-block list, or blocks missing from the file system. If the `-q` option is specified, the free-block list will be salvaged automatically.

Possible responses to SALVAGE prompt are:

- | | |
|-----|---|
| YES | Replace actual free-block list with a new free-block list. The new free-block list will be ordered to reduce time spent by the disk waiting for the disk to rotate into position. |
| NO | Ignore this error condition. |

I. PHASE 6: SALVAGE FREE LIST

This phase concerns itself with the free-block list reconstruction. This part lists error conditions resulting from the blocks-to-skip and blocks-per-cylinder values.

Default free-block list spacing assumed

This is an advisory message indicating the blocks-to-skip is greater than the blocks-per-cylinder, the blocks-to-skip is less than 1, the blocks-per-cylinder is less than 1, or the blocks-per-cylinder is greater than 500. The default values of 9 blocks-to-skip and 400 blocks-per-cylinder are used. See **fsck(1M)** in section 1 of the Runtime System manual for further details.

J. CLEANUP

Once a file system has been checked, a few cleanup functions are performed. This part lists advisory messages about the file system and modify status of the file system.

X files Y blocks Z free

This is an advisory message indicating that the file system checked contained *X* files using *Y* blocks leaving *Z* blocks free in the file system.

******* BOOT UNIX (NO SYNC!) *******

This is an advisory message indicating that a mounted file system or the root file system has been modified by **fsck**. If the UNIX system is not rebooted immediately without **sync**, the work done by **fsck** may be undone by the in-core copies of tables the UNIX system keeps.

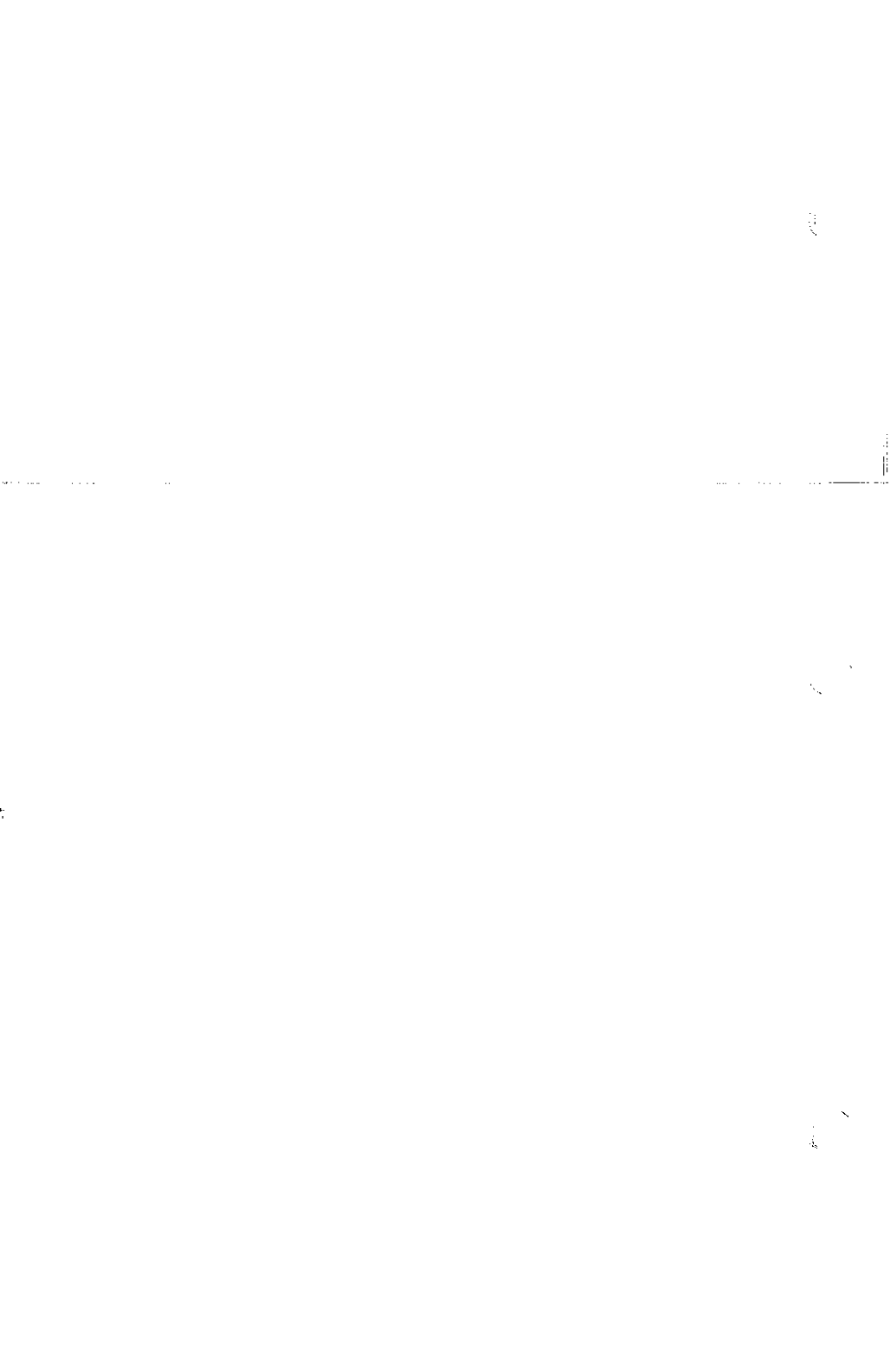
******* FILE SYSTEM WAS MODIFIED *******

This is an advisory message indicating that the current file system was modified by **fsck**.

Chapter 10

LP SPOOLING SYSTEM

	PAGE
GENERAL	1
OVERVIEW OF LP FEATURES	2
Definitions.....	2
Commands.....	3
LP SCHEDULER STARTUP	5
CONFIGURING LP—THE “lpadmin” COMMAND	5
Introducing New Destinations.....	5
Modifying Existing Destinations.....	7
Specifying the System Default Destination.....	9
Removing Destinations.....	9
MAKING AN OUTPUT REQUEST—THE “lp” COMMAND	10
FINDING LP STATUS—LPSTAT	12
CANCELING REQUESTS—CANCEL	12
ALLOWING AND REFUSING REQUESTS—ACCEPT AND REJECT	13
ALLOWING AND INHIBITING PRINTING—ENABLE AND DISABLE	14
MOVING REQUESTS BETWEEN DESTINATIONS	
LP MOVE	15
STOPPING AND STARTING THE SCHEDULER—LP SHUT AND LPSCHED	16
PRINTER INTERFACE PROGRAMS	17
SETTING UP HARD-WIRED DEVICES AND LOGIN	
TERMINALS AS LP PRINTERS	19
Hard-wired Devices.....	19
Login Terminals.....	21
SUMMARY	22



LP SPOOLING SYSTEM

GENERAL

The line printer (LP) program is a series of commands that perform diverse spooling functions under the UNIX operating system. Since the primary LP application is off-line printing, this document focuses mainly on spooling to line printers. LP allows administrators to customize the system to spool to a collection of line printers of any type and to group printers into logical classes in order to maximize the throughput of the devices. Users are provided the capabilities of:

- Queuing and canceling print requests
- Preventing and allowing queuing to devices
- Starting and stopping LP from processing requests
- Changing configuration of printers
- Finding status of the LP system.

This chapter describes the role of an LP administrator in performing restricted functions and overseeing the smooth operation of LP.

Throughout this chapter, each reference of the form **name(N)**, where "N" is the number 1 or 7 possibly followed by a letter, refer to entry **name** in section N of the Runtime System manual. If "N" is a number 2 through 5 possibly followed by a letter, refer to entry **name** in section N of the Software Development System manual.

OVERVIEW OF LP FEATURES

Definitions

Several terms must be defined before presenting a brief summary of LP commands. The LP was designed with the flexibility to meet the needs of users on different UNIX systems. Changes to the LP configuration are performed by the **lpadmin(1M)** command.

LP makes a distinction between printers and printing devices. A *device* is a physical peripheral device or a file and is represented by a full UNIX system pathname. A *printer* is a logical name that represents a device. At different points in time, a printer may be associated with different devices. A *class* is a name given to an ordered list of printers. Every class must contain at least one printer. Each printer may be a member of zero or more classes. A *destination* is a printer or a class. One destination may be designated as the *system default destination*. The **lp(1)** command will direct all output to this destination unless the user specifies otherwise. Output that is routed to a printer will be printed only by that printer, whereas output directed to a class will be printed by the first available class member.

Each invocation of **lp** creates an output request that consists of the files to be printed and options from the lp command line. An interface program which formats requests must be supplied for each printer. The LP scheduler, **lpsched(1M)**, services requests for all destinations by routing requests to interface programs to do the printing on devices. An LP configuration for a system consists of devices, destinations, and interface programs.

Commands

Commands for General Use

The **lp(1)** command is used to request the printing of files. It creates an output request and returns a request id of the form

`dest-seqno`

to the user, where `seqno` is a unique sequence to that printer, *dest-seqno* is a unique sequence number across the entire LP system, and *dest* is the destination where the request was routed.

Cancel is used to cancel output requests. The user supplies request ids as returned by **lp** or printer names, in which case the currently printing requests on those printers are canceled.

Disable prevents **lpsched** from routing output requests to printers.

Enable(1) allows **lpsched** to route output requests to printers.

Commands for LP Administrators

Each LP system must designate a person or persons as LP administrator to perform the restricted functions listed below. Either the superuser or any user who is logged into the UNIX system as **lp** qualifies as an LP administrator. All LP files and commands are owned by **lp** except for **lpadmin** and **lpsched** which are owned by root. The following commands will be described in more detail later in this chapter.

lpadmin(1M)	Modifies LP configuration. Many features of this command cannot be used when lpsched is running.
lpsched(1M)	Routes output requests to interface programs which do the printing on devices.
lpshut	Stops lpsched from running. All printing activity is halted, but other LP commands may still be used.
accept(1M)	Allows lp to accept output requests for destinations.
reject	Prevents lp from accepting requests for destinations.
lpmove	Moves output requests from one destination to another. Whole destinations may be moved at one time. This command cannot be used when lpsched is running.

LP SCHEDULER STARTUP

You should find the following code in */etc/rc.d/lp.rc*:

```
rm -f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
echo "LP scheduler started"
```

This starts the LP scheduler each time that the UNIX system is restarted.

PRECAUTIONS

1. Some LP commands invoke other LP commands. Moving them after they are installed will cause some commands to fail.
2. The files under the SPOOL directory should be modified **only by LP commands**.
3. All LP commands require set-user-id permission. If this is removed, the commands will fail.

CONFIGURING LP—THE “lpadmin” COMMAND

Changes to the LP configuration should be made by using the **lpadmin** command and not by hand. **lpadmin** will not attempt to alter the LP configuration when **lpsched** is running, except where explicitly noted below.

Introducing New Destinations

The following information must be supplied to **lpadmin** when introducing a new printer:

1. The printer name (**-p printer**) is an arbitrary name which must conform to the following rules:
 - It must be no longer than 14 characters.
 - It must consist solely of alphanumeric characters and underscores.
 - It must not be the name of an existing LP destination (printer or class).

LP SPOOLING

2. The device associated with the printer (`-v device`). This is the path name of a hard-wired printer, a login terminal, or other file that is writable by `lp`.
3. The printer interface program. This may be specified in one of three ways:
 - It may be selected from a list of model interfaces supplied with `LP -m model`.
 - It may be the same interface that an existing printer uses (`-e printer`).
 - It may be a program supplied by the LP administrator (`-i interface`).

Information which need not always be supplied when creating a new printer includes:

1. The user may specify `-h` to indicate that the device for the printer is hardwired or the device is the name of a file (this is assumed by default). If, on the other hand, the device is the path name of a login terminal, then `-I` must be included on the command line. This indicates to `lpsched` that it must automatically disable this printer each time `lpsched` starts running. This fact is reported by `lpstat` when it indicates printer status:

```
$ lpstat -pa  
printer a (login terminal) disabled Oct 31 11:15 -  
disabled by scheduler: login terminal
```

This is done because device names for login terminals can be (and usually are) associated with different physical devices from day to day. If the scheduler did not take this action, somebody might log in and be surprised that LP is spooling to his/her terminal!

2. The new printer may be added to an existing class or added to a new class (`-c class`). New class names must conform to the same rules for new printer names.

EXAMPLES

The following examples will be referenced by further examples in later sections.

1. Create a printer called `pr1` whose device is `/dev/printer` and whose interface program is the model `hp` interface:

```
$ /usr/lib/lpadmin -ppr1 -v/dev/printer -mhp
```

2. Add a printer called `pr2` whose device is `/dev/tty22` and whose interface is a variation of the model `prx` interface. It is also a login terminal:

```
$ cp /usr/spool/lp/model/prx xxx
  < edit xxx >
$ /usr/lib/lpadmin -ppr2 -v/dev/tty22 -ixxx -l
```

3. Create a printer called `pr3` whose device is `/dev/tty23`. The `pr3` will be added to a new class called `cl1` and will use the same interface as printer `pr2`:

```
$ /usr/lib/lpadmin -ppr3 -v/dev/tty23 -ep2 -ccl1
```

Modifying Existing Destinations

Modifications to existing destinations must always be made with respect to a printer name (`-pprinter`). The modifications may be one or more of the following:

1. The device for the printer may be changed (`-vdevice`). If this is the only modification, then this may be done even while `lpsched` is running. This facilitates changing devices for login terminals.
2. The printer interface program may be changed (`-mmodel`, `-eprinter`, `-iinterface`).
3. The printer may be specified as hardwired (`-h`) or as a login terminal (`-l`).

4. The printer may be added to a new or existing class (`-cclass`).
5. The printer may be removed from an existing class (`-rclass`). Removing the last remaining member of a class causes the class to be deleted. No destination may be removed if it has pending requests. In that case, `lpmove` or `cancel` should be used to move or delete the pending requests.

EXAMPLES

These examples are based on the LP configuration created by those in the previous section.

1. Add printer `pr2` to class `cl1`:

```
$ /usr/lib/lpadmin -ppr2 -ccl1
```

2. Change `pr2`'s interface program to the model `prx` interface, change its device to `/dev/tty24`, and add it to a new class called `cl2`:

```
$ /usr/lib/lpadmin -ppr2 -mprx -v/dev/tty24 -ccl2
```

Note that printers `pr2` and `pr3` now use different interface programs even though `pr3` was originally created with the same interface as `pr2`. Printer `pr2` is now a member of two classes.

3. Specify printer `pr2` as a hard-wired printer:

```
$ /usr/lib/lpadmin -ppr2 -h
```

4. Add printer `pr1` to class `cl2`:

```
$ /usr/lib/lpadmin -ppr1 -ccl2
```

The members of class `cl2` are now `pr2` and `pr1`, in that order. Requests routed to class `cl2` will be serviced by `pr2` if both `pr2` and `pr1` are ready to print; otherwise, they will be printed by the one which is next ready to print.

- Remove printers `pr2` and `pr3` from class `cl1`:

```
$ /usr/lib/lpadmin -ppr2 -rcl1
$ /usr/lib/lpadmin -ppr3 -rcl1
```

Since `pr3` was the last remaining member of class `cl1`, the class is removed.

- Add `pr3` to a new class called `cl3`.

```
$ /usr/lib/lpadmin -ppr3 -ccl3
```

Specifying the System Default Destination

The system default destination may be changed even when `lpsched` is running.

EXAMPLES

- Establish class `cl1` as the system default destination:

```
$ /usr/lib/lpadmin -dcl1
```

- Establish no default destination:

```
$ /usr/lib/lpadmin -d
```

Removing Destinations

Classes and printers may be removed only if there are no pending requests that were routed to them. Pending requests must either be canceled using `cancel` or moved to other destinations using `lpmove` before destinations may be removed. If the removed destination is the system default destination, then the system will have no default destination until the default destination is respecified. When the last remaining member of a class is removed, then the class is also removed. The removal of a class never implies the removal of printers.

EXAMPLES

1. Make printer pr1 the system default destination:

```
$ /usr/lib/lpadmin -dpr1
```

Remove printer pr1:

```
$ /usr/lib/lpadmin -xpr1
```

Now there is no system default destination.

2. Remove printer pr2:

```
$ /usr/lib/lpadmin -xpr2
```

Class cl2 is also removed since pr2 was its only member.

3. Remove class cl3:

```
$ /usr/lib/lpadmin -xcl3
```

Class cl3 is removed, but printer pr3 remains.

MAKING AN OUTPUT REQUEST— THE “lp” COMMAND

Once LP destinations have been created, users may request output by using the **lp** command. The request id that is returned may be used to see if the request has been printed or to cancel the request.

The LP program determines the destination of a request by checking the following list in order:

- If the user specifies *-ddest* on the command line, then the request is routed to *dest*.

- If the environment variable **LPDEST** is set, the request is routed to the value of *LPDEST*.
- If there is a system default destination, then the request is routed there.
- The request is rejected.

EXAMPLES

1. There are at least four ways to print the password file on the system default destination:

```
lp /etc/passwd
lp < /etc/passwd
cat /etc/passwd | lp
lp -c /etc/passwd
```

All four methods produce a printed copy of the file. The last method creates a *duplicate* of the file and then prints a copy of the duplicate. The duplicate file is removed after the print job is completed. Thus, with the first three methods if the file is modified between the time the request is made and the time it is actually printed, then the changes will be reflected in the output.

2. Print two copies of file abc on printer xyz and title the output "my file":

```
pr abc | lp -dxyz -n2 -t" my file"
```

3. Print file xxx on a Diablo* 1640 printer called zoo in 12-pitch and write to the user's terminal when printing has completed:

```
lp -dzoo -o12 -w xxx
```

* Registered trademark of Xerox Corporation

In this example, "12" is an option that is meaningful to the model Diablo 1640 interface program that prints output in 12-pitch mode [see `lpadmin(1M)`].

FINDING LP STATUS—LPSTAT

The `lpstat` command is used to find status information about LP requests, destinations, and the scheduler.

EXAMPLES

1. List the status of all pending output requests made by this user:

```
lpstat
```

The status information for a request includes the request id, the log name of the user, the total number of characters to be printed, and the date and time the request was made.

2. List the status of printers p1 and p2:

```
lpstat -pp1,p2
```

CANCELING REQUESTS—CANCEL

The LP requests may be canceled using the `cancel` command. Two kinds of arguments may be given to the command—request ids and printer names. The requests named by the request ids are canceled and requests that are currently printing on the named printers are canceled. Both types of arguments may be intermixed.

EXAMPLE

Cancel the request that is now printing on printer xyz:

```
cancel xyz
```

If the user that is canceling a request is not the same one that made the request, then mail is sent to the owner of the request. LP allows any user to cancel requests in order to eliminate the need for users to

find LP administrators when unusual output should be purged from printers.

ALLOWING AND REFUSING REQUESTS— ACCEPT AND REJECT

When a new destination is created, **lp** will reject requests that are routed to it. When the LP administrator is sure that it is set up correctly, he or she should allow **lp** to accept requests for that destination. The **accept** command performs this function.

Sometimes it is necessary to prevent **lp** from routing requests to destinations. If printers have been removed or are waiting to be repaired or if too many requests are building for printers, then it may be desirable to cause **lp** to reject requests for those destinations. The **reject** command performs this function. After the condition that led to the rejection of requests has been remedied, the **accept** command should be used to allow requests to be taken again.

The acceptance status of destinations is reported by the **-a** option of **lpstat**.

EXAMPLES

1. Cause **lp** to reject requests for destination xyz:

```
/usr/lib/reject -r" printer xyz needs repair" xyz
```

Any users that try to route requests to xyz will encounter the following:

```
$ lp -dxyz file
lp: can not accept requests for destination "xyz"
-- printer xyz needs repair
```

2. Allow **lp** to accept requests routed to destination **xyz**:

```
/usr/lib/accept xyz
```

ALLOWING AND INHIBITING PRINTING— ENABLE AND DISABLE

The **enable** command allows the LP scheduler to print requests on printers. That is, the scheduler routes requests only to the interface programs of enabled printers. Note that it is possible to enable a printer and at the same time prevent further requests from being routed to it.

The **disable** command will undo the effects of the **enable** command. It prevents the scheduler from routing requests to printers, independently of whether or not **lp** is allowing them to accept requests. Printers may be disabled for several reasons including malfunctioning hardware, paper jams, and end of day shutdowns. If a printer is busy at the time it is disabled, then the request that was printing will be reprinted in its entirety either on another printer (if the request was originally routed to a class of printers) or on the same one when the printer is reenabled. The **-c** option causes the currently printing requests on busy printers to be canceled in addition to disabling the printers. This is useful if strange output is causing a printer to behave abnormally.

EXAMPLE

Disable printer **xyz** because of a paper jam:

```
$ disable -r" paper jam" xyz  
printer " xyz" now disabled
```

Find the status of printer **xyz**:

```
$ lpstat -pxyz  
printer " xyz" disabled since Jan 5 10:15 -  
paper jam
```

Now, reenable xyz:

```
$ enable xyz
printer "xyz" now enabled
```

MOVING REQUESTS BETWEEN DESTINATIONS—LPMOVE

Occasionally, it is useful for LP administrators to move output requests between destinations. For instance, when a printer is down for repairs, it may be desirable to move all of its pending requests to a working printer. This is one way to use the **lpmove** command. The other use of this command is to move specific requests to a different destination. **lpmove** will refuse to move requests while the LP scheduler is running.

EXAMPLES

1. Move all requests for printer abc to printer xyz:

```
$ /usr/lib/lpmove abc xyz
```

All of the moved requests are renamed from abc-*nnn* to xyz-*nnn*. As a side effect, destination abc is no longer accepting further requests.

2. Move requests zoo-543 and abc-1200 to printer xyz:

```
$ /usr/lib/lpmove zoo-543 abc-1200 xyz
```

The two requests are now renamed xyz-543 and xyz-1200.

STOPPING AND STARTING THE SCHEDULER—LPSHUT AND LPSCHED

Lpsched is the program that routes the output requests that were made with **lp** through the appropriate printer interface programs to be printed on line printers. Each time the scheduler routes a request to an interface program, it records an entry in the log file, */usr/spool/lp/log*. This entry contains the log name of the user that made the request, the request id, the name of the printer that the request is being printed on, and the date and time that printing first started. In the case that a request has been restarted, more than one entry in the log file may refer to the request. The scheduler also records error messages in the log file. When **lpsched** is started, it renames */usr/spool/lp/log* to */usr/spool/lp/oldlog* and starts a new log file.

No printing will be performed by the LP system unless **lpsched** is running. Use the command

```
lpstat -r
```

to find the status of the LP scheduler.

Lpsched is normally started by the */etc/rc.d/lp.rc* program as described above and continues to run until the UNIX system is shut down. The scheduler operates in the */usr/spool/lp* directory. When it starts running, it will exit immediately if a file called *SCHEDLOCK* exists. Otherwise, it creates this file in order to prevent more than one scheduler from running at the same time.

Occasionally, it is necessary to shut down the scheduler in order to reconfigure LP or to rebuild the LP software. The command

```
/usr/lib/lpshut
```

causes **lpsched** to stop running and terminates all printing activity. All requests that were in the middle of printing will be reprinted in their entirety when the scheduler is restarted.

To restart the LP scheduler, use the command

```
/usr/lib/lpsched
```

Shortly after this command is entered, **lpstat** should report that the scheduler is running. If not, it is possible that a previous invocation of **lpsched** exited without removing **SCHEDLOCK**, so try the following:

```
rm -f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
```

The scheduler should be running now.

PRINTER INTERFACE PROGRAMS

Every LP printer must have an interface program which does the actual printing on the device that is currently associated with the printer. Interface programs may be shell procedures, C programs, or any other executable program. The LP model interfaces are all written as shell procedures and can be found in the */usr/spool/lp/model* directory. At the time **lpsched** routes an output request to a printer P, the interface program for P is invoked in the directory */usr/spool/lp* as follows:

```
interface/P id user title copies options file ...
where
id is the request id returned by lp
user is log name of user who made the request
title is optional title specified by the user
copies is number of copies requested by user
options is a blank-separated list of class or
printer-dependent options specified by user
file is the full path name of a file to be printed
```

EXAMPLES

The following examples are requests made by user "smith" with a system default destination of printer "xyz". Each example lists an **lp** command line followed by the corresponding command line generated for printer xyz's interface program:

1. `lp /etc/passwd /etc/group
interface/xyz xyz-52 smith " " 1 " " /etc/passwd /etc/group`
2. `pr /etc/passwd | lp -t" users" -n5
interface/xyz xyz-53 smith users 5 ""
/usr/spool/lp/request/xyz/d0-53`
3. `lp /etc/passwd -oa -ob
interface/xyz xyz-54 smith " " 1 " a b" /etc/passwd`

When the interface program is invoked, its standard input comes from /dev/null and both the standard output and standard error output are directed to the printer's device. Devices are opened for reading as well as writing when file modes permit. In the case where a device is a regular file, all output is appended to the end of the file.

Given the command line arguments and the output directed to a device, interface programs may format their output in any way they choose. On *serial* printers, interface programs must ensure that the proper stty modes (terminal characteristics such as baud rate, output options, etc.) are in effect on the output device. This may be done in a shell interface only if the device is opened for reading:

```
stty mode ... <&1
```

That is, take the standard input for the stty command from the device. Stty commands are not applicable to parallel printers.

When printing has completed, it is the responsibility of the interface program to exit with a code indicative of the success of the print job. Exit codes are interpreted by **lpsched** as follows.

CODE	MEANING TO LPSCHED
0	The print job has completed successfully.
1 to 127	A problem was encountered in printing this particular request (e.g., too many nonprintable characters). This problem will not affect future print jobs. Lpsched notifies users by mail that there was an error in printing the request.
greater than 127	These codes are reserved for internal use by lpsched . Interface programs must not exit with codes in this range.

When problems that are likely to affect future print jobs occur (e.g., a device filter program is missing), the interface programs would be wise to disable printers so that print requests are not lost. When a busy printer is disabled, the interface program will be terminated with signal 15.

SETTING UP HARD-WIRED DEVICES AND LOGIN TERMINALS AS LP PRINTERS

C-10

Hard-wired Devices

As an example of how to set up a hard-wired device for use as an LP printer, consider using tty line 15 as printer xyz. As superuser, perform the following:

1. Avoid unwanted output from non-LP processes and ensure that LP can write to the device:

```
$ chown lp /dev/tty15
$ chmod 600 /dev/tty15
```

2. Change */etc/inittab* so that tty15 is not a login terminal. In other words, ensure that */etc/getty* is not trying to log users in at

this terminal. Change the entries for `tty15` to:

```
15:2:off:/etc/getty -t60 tty15 1200
```

Enter the command:

```
$ telinit Q
```

If there is currently an invocation of `/etc/getty` running on `tty15`, kill it. When the UNIX system is rebooted, `tty15` will be initialized with default stty modes. Thus, it is up to LP interface programs to establish the proper baud rate and other stty modes for correct printing to occur.

3. Introduce printer `xyz` to LP using the model `prx` interface program:

```
$ /usr/lib/lpadmin -pxyz -v/dev/tty15 -mprx
```

4. When `xyz` is created, it will initially be disabled and `lp` will be rejecting requests routed to it. If it is desired, allow `lp` to accept requests for `xyz`:

```
/usr/lib/accept xyz
```

This will allow requests to build up for `xyz` and to be printed when it is enabled at a later time.

5. When it is desired for printing to occur, be sure that the printer is ready to receive output. For some printers, this means that the top of form has been adjusted and that the printer is on-line. Enable printing to occur on `xyz`:

```
enable xyz
```

When requests have been routed to `xyz`, they will begin printing.

Login Terminals

Login terminals may also be used as LP printers. To do this for a Diablo 1640 terminal called *abc*, perform the following:

1. Introduce printer *abc* to LP using the model 1640 interface program:

```
$ /usr/lib/lpadmin -pabc -v/dev/null -m1640 -l
```

Note that */dev/null* is used as *abc*'s device because we will specify the actual device each time that *abc* is enabled. This device may be different from day to day. When *abc* is created, it will initially be disabled; and **lp** will be rejecting requests routed to it. If it is desired, allow **lp** to accept requests for *abc*:

```
/usr/lib/accept abc
```

This will allow requests to build up for *abc* and to be printed when it is enabled at a later time. It is not advisable to enable *abc* for printing, however, until the following steps have been taken.

2. Log terminal in if this has not already been done.
3. Assuming the **tty(1)** command reports that this terminal is */dev/tty02*, associate this device with printer *abc*:

```
$ /usr/lib/lpadmin -pabc -v/dev/tty02
```

Note that **lpadmin** may be used only by an LP administrator. If it is desired for other users to routinely perform this step, then an LPA may establish a program owned by **lp** or by **root** with set-user-id permission that performs this function.

4. When it is desired for printing to occur, be sure that the printer is ready to receive output. For some printers, this means that the top of form has been adjusted. Enable printing to occur on abc:

```
enable abc
```

When requests have been routed to abc, they will begin printing.

5. When all printing has stopped on abc or when you want it back as a regular login terminal, you may prevent it from printing more output:

```
$ disable abc  
printer " abc" now disabled
```

If abc is enabled when the UNIX system is rebooted or when **lpsched** is restarted, it will be disabled automatically.

SUMMARY

The administrative functions of the LP administrator have been described in detail. These functions include configuring and reconfiguring LP; maintaining printer interface programs; accepting, rejecting, and moving print requests; stopping and starting the LP scheduler; and enabling and disabling printers. LP offers administrators the following advantages over other centrally supported printer packages:

- Printers may be grouped into classes.
- LP may be configured to meet the needs of each site.
- Administrators may supply interface programs to format output in any way desirable.
- LP functions are performed by simple commands and not by hand.

Chapter 11

COMMUNICATION TUTORIAL

	PAGE
INTRODUCTION	1
COMMUNICATING ON THE UNIX SYSTEM	2
HOW CAN YOU COMMUNICATE?.....	3
SENDING AND RECEIVING MESSAGES (<i>mail</i>).....	4
Sending Mail.....	4
Basics of Sending Mail.....	4
Sending Mail to One Person.....	5
Sending Mail to Several People Simultaneously	7
Sending Mail to Remote Systems (<i>uname, uuname</i>)	8
Receiving Mail.....	12
SENDING AND RECEIVING FILES.....	17
Sending Small Files (<i>mail</i>)	17
Sending Large Files (<i>uuto</i>).....	19
Have You Got Permission?.....	19
Sending a File (<i>uuto -m. uustat</i>)	21
Receiving Files (<i>uupick</i>)	26
ADVANCED MESSAGE AND FILE HANDLING (<i>uucp, mailx</i>)	29



COMMUNICATION TUTORIAL

INTRODUCTION

Sooner or later, you will want to use the UNIX system to get in touch with other UNIX system users. You may want to send a message to someone; the message may be one that must be read immediately. Perhaps you might need to send another user information from a file in your login.

Whatever the case, this chapter teaches you how to use the communication tools available to you on the UNIX system. The chapter begins with a brief overview of just who you might want to communicate with on the UNIX system. You learn how to send basic messages to users on your system and other UNIX systems, and also how to deal with messages you receive. You also learn about commands that enable you to send files to other users.

The following list is a review of the text conventions that are used in this chapter:

- bold** (Commands typed in exactly as shown.)
- italic* (UNIX system prompts and responses.)
- roman (Input other than commands.)
- <> (Commands that are typed in, but are not reflected on the screen, are enclosed in angle brackets.)

COMMUNICATING ON THE UNIX SYSTEM

You can use the UNIX system to communicate with just about anyone else who uses the UNIX system. This means that your terminal does more than serve as a work station--it becomes your personal message-handling center as well, with the electronic equivalent of transmission, routing, and storage facilities.

Who would you want to communicate with over the UNIX system? Here are some examples to consider:

- The person in your office who needs to know about a department meeting tomorrow,
- Other users on your UNIX system who should see a posted message concerning their use of the system after office hours,
- The supervisor who wants a copy of your last two reports by 2:30 this afternoon,
- The supervisor who wants to review the memo you are presently working on as soon as you have finished it,
- A person working with you on the UNIX system to modify several files you both have in common; you need to be in touch from time to time, but the phones are being used as links from your terminals to the computer and you would rather not shout down the halls, and
- A coordinator who wants your daily operations records (all in very large files), but does not want to have to wade through them all at once when he receives them on the terminal.

As you can see, you can keep in touch with any number of people for any number of reasons through the UNIX system. The remainder of this chapter shows you how to use the various communication tools provided by the UNIX system to reach these people.

HOW CAN YOU COMMUNICATE?

The UNIX system offers several commands for user-to-user communication. This chapter explains the most important commands to know and suggests how to select the one to use in a given situation. The basic choice is between sending (or receiving) a message and sending (or receiving) a file.

To expand on one of the previous examples, suppose you are working at your terminal and you remember that you are giving a presentation at an officewide meeting tomorrow. You want to remind someone in your office about the presentation, but you do not want to take the time for a phone call or a walk to the other person's office. What can you do?

If the other person has a login on your UNIX system, you can use the **mail** command to send a brief message. When the recipient of your message finishes whatever task he or she is using the UNIX system for, a notice is posted that there is mail waiting to be read. The recipient can then read your message and send a reply back to your login.

To take another example, what if you need to send other people copies of things you already have on file--memos, reports, saved messages, documents, and the like? You can send such files using the **mail** command; however, this may not be the best way to send long files. For sending files over a page in length, you should use the **uuto** command. This command sends the file to a public directory on the recipient's system instead of sending it straight to the recipient's login. The recipient can then deal with it at his or her own leisure.

These are the important communication tools available to you. (Two other tools, the **uucp** and **mailx** commands, are discussed briefly at the end of the chapter.) Now that you have a general idea of how to communicate in the UNIX system, let's move on to the specifics.

SENDING AND RECEIVING MESSAGES (*mail*)

The **mail** command works in two ways--it lets you send messages to other UNIX system users, and it lets you read messages sent to you. This section deals first with sending messages, both to users on your UNIX system and to users on other UNIX systems that can communicate with yours.

Sending Mail

It is easy to send mail to another user. The basic command line format for sending mail is

```
mail login<CR>
```

where *login* is the recipient's login name on the UNIX system. This login name can be either of the following:

- A login name if the recipient is on your system, or
- A system name and login name if the recipient is on a system that can communicate with yours.

For the moment, assume that the recipient is on your system (known as the local system); we will deal with sending mail to users on other systems (known as remote systems) a little later.

Basics of Sending Mail

Since the recipient is on your system, you type the **mail** command as follows at the system prompt (\$):

```
mail login<CR>  
text
```

where *login* is the recipient's login name. Then you type in the text of the letter, as many lines as you need. When your message is complete, you send the message on its way by typing a dot (.) at the beginning of a new line.

The resulting message looks like this:

```
$ mail login<CR>
After you enter the command line,<CR>
type in as many lines of text as you need<CR>
to get the message across.<CR>
When you're done,<CR>
type in a control-d or a dot<CR>
on a line by itself, as shown on the next line.<CR>
.<CR>
$
```

The system prompt returns to notify you that your message has been queued (placed in line) and will be sent.

Sending Mail to One Person

Let's look at a sample situation. You have to notify another person in your office of a meeting later this afternoon, but he is not in and you have to leave your office. He has a login on your UNIX system with the login name *tommy*, so you can leave a message for him to read the next time he logs into the system:

```
$ mail tommy<CR>
Tom,<CR>
There's a meeting of the review committee<CR>
at 3:00 this afternoon. D.F. wants your<CR>
comments and an idea of how long you think<CR>
the project will take to complete.<CR>
B.K.<CR>
.
$
```

COMMUNICATION TUTORIAL

When Tom logs in at his terminal (or while he is already logged in), he receives a message that tells him he has mail waiting:

```
you have mail
```

To see how *tommy* can read his mail, see the section titled *Receiving Mail*.

You can practice using the **mail** command by sending mail to yourself. This may sound strange at first, but it is the easiest way to practice sending messages. Simply type in the **mail** command and your own login name, then write a short message to yourself. When you type in the dot, the mail will be sent to your login and you will receive the notice that you have mail.

Sending mail to yourself can also serve as a handy reminder system. Suppose your login name is *rover*; you are ready to log off of the system for the day and you want to leave a reminder to call someone first thing the next morning. You might enter the following:

```
$ mail rover<CR>  
Remember to call Accounting and find out<CR>  
why they haven't returned my 1984 figures!<CR>  
.  
$
```

When you log in the next day, you will get a notice of messages awaiting you. Reading your mail then brings up the reminder message (and any other messages you may have received).

Sending Mail to Several People Simultaneously

If you need to send the same message to more than one person, simply place their login names after the **mail** command on the command line, with a space between each one, in the following format:

```
mail login1 login2 ... <CR>
```

where *login1*, *login2*, and ... are the different login names. You can mail messages to as many logins as you wish.

For example, if you send a notice about the department softball game to team members with login names *tommy*, *switch*, *wombat*, and *dave*, it might look like this:

```
$ mail tommy switch wombat dave<CR>
Diamond cutters,<CR>
The game is on for tonight at diamond three.<CR>
Don't forget your gloves!<CR>
Your Manager<CR>
.<CR>
$
```

To provide you with a quick summary of what you can expect when using the **mail** command to send messages, a recap of how to use it follows.

Command Recap

mail - sends a message to another user's login

<i>command</i>	<i>options</i>	<i>arguments</i>
mail	none	login

Description: **mail** followed by one or more login names, sends the message typed on the lines following the command line to the specified login(s).

Remarks: Typing a dot at the beginning of a new line sends the message.

Sending Mail to Remote Systems (*uname, uuname*)

We have assumed to this point that you are sending messages to recipients on your (local) UNIX system. You may have occasion, however, to send messages to recipients on other (remote) UNIX systems. For example, your office may have three separate systems, each in a different part of the building. Or perhaps you may have offices in several different locations, each with its own system.

How do you send mail to someone on a remote system? The UNIX system you are on must be able to communicate with a remote UNIX system before mail can be sent between the two. So, if you plan to send a mail message to someone on a remote system, you need to do a little legwork to find out the following information:

- Recipient's login name,
- Name of the remote system, and
- If your system and the remote system can communicate.

Two commands are available to help you answer these questions--the **uname** and **uuname** commands.

You can get the login name and the remote system name from the recipient. If it happens that the recipient does not remember the system name, have him or her log into the system and type the following at the system prompt:

```
uname -n<CR>
```

The **uname -n** command responds with the name of the system you are logged into. For example, if you are logged into a system named *sys10* and you type in **uname -n**, your screen should look like this:

```
$ uname -n<CR>
sys10
$
```

Once you know the remote system name, the **uuname** command helps you find out if your system can communicate with the remote system. At the prompt, type:

```
uuname<CR>
```

This generates a list containing the names of remote systems with which your system can communicate. If the recipient's system is in that list, then you can send messages there by **mail**.

The **uuname** command may respond with a large list of names if your system can communicate with many other systems. To avoid having that long list scroll quickly up your screen, use the pipe and **grep** command in conjunction with **uuname**. At the prompt, type:

```
uuname | grep system<CR>
```

where *system* is the recipient's system name. This generates the same list, then searches for and prints only the specified system name if it is found in the list.

For example, if you want to find out whether a system called *sys10* can communicate with your system, type:

```
$ uname | grep sys10<CR>
```

If this is the case, the system name is printed in response:

```
$ uname | grep sys10<CR>
sys10
$
```

If you get only the system prompt back, then the two systems cannot communicate:

```
$ uname | grep sys10<CR>
$
```

Once you determine that you can send messages to a login on a remote system, your **mail** command line is slightly different than it is for sending mail to someone on your local system. The command line format for remote systems is:

```
mail system!login<CR>
```

where *system* is the remote system name and *login* is the recipient's login name. The two parts of the address are separated by an exclamation point (!).

Now that you have all the parts, let's put them together into an example. Assume that you have a message for someone on a different system in another part of your office. You know from the recipient her login name, *sarah*, and her system name, *sys10*. To find out if her system can communicate with yours, use the **uname** command:

```
$ uname | grep sys10<CR>
sys10
$
```

The system response tells you that your system is indeed networked to system *sys10*. Now all you have to do is send the message, using the expanded address format given previously:

```
$ mail sys10!sarah<CR>
Sarah,<CR>
The final counts for the writing seminar<CR>
are as follows:<CR>
<CR>
Our department - 18<CR>
Your department - 20<CR>
<CR>
Tom<CR>
.<CR>
$
```

Following is a quick summary of the two commands introduced in this section and what you can expect them to do.

Command Recap

uname - displays the system name

<i>command</i>	<i>options</i>	<i>arguments</i>
uname	-n and others*	none

Description: **uname -n** displays the name of the system on which your login resides.

* See the *UNIX System User Reference Manual* for all available options and an explanation of their capabilities.

Command Recap

uuname - displays a list of networked systems

<i>command</i>	<i>options</i>	<i>arguments</i>
uuname	none	none

Description: **uuname** displays a list of remote systems that can communicate with your system.

Receiving Mail

Once you learn to send messages, you may be anxious to read what others are sending your way. As stated earlier, the **mail** command also allows you to read messages sent by other UNIX system users.

After logging in, you may receive the following message at your terminal:

```
you have mail
```

This tells you that one or more messages are being held for you in a UNIX system directory named *usr/mail*, usually referred to as the *mailbox*. Entering the **mail** command by itself allows you to read these messages.

To read your mail, type the **mail** command by itself at the system prompt:

```
mail<CR>
```

This displays the waiting messages at your terminal, one message at a time, with the most recently received message displayed first. In other words, as you read your messages, you go from the "newest" message to the "oldest" message.

A typical **mail** message looks like this:

```
$ mail
From tommy Mon May 21 15:33 CST 1984
B.K.
Looks like the meeting has been canceled.
Do you still want the technical review material?
Tom
?
```

COMMUNICATION TUTORIAL

The first line, called the *header*, displays information about a particular message--the login name of the sender, the date sent, and the time sent. The following lines (except for the last line) are the body of the message.

Notice the question mark (?) on the last line of the message. After displaying each message, the **mail** command displays a ? and a space, and waits for a response from you before going on to the next message. There are several responses; we will look at the most common responses and what they do.

After reading a message, you may want to delete it. To do so, type a **d** after the question mark.

```
? d<CR>
```

This response deletes the message from the *mailbox* and displays the next message waiting in the *mailbox* (if there is one). If there are no other messages, the system prompt returns to indicate that you've finished reading your messages.

If you would rather display the next message without deleting the message being displayed, type a carriage return after the question mark.

```
? <CR>
```

The current message goes back into the *mailbox* and the next message is displayed. If there are no more messages in the *mailbox*, the system prompt returns.

You may want to save the message for later reference. To do so, type an **s** after the question mark:

```
? s<CR>
```

This response saves the mail message by default in a file called *mbx* in your login directory. If you would rather save the message in another file, follow the **s** response with a file name or with a path name ending in a file name.

For example, to save the message in a file called *mailsave* in your current directory, enter the following response after the question mark:

```
? s mailsave<CR>
```

If you use the **ls** command to list the contents of this directory, you will find the file *mailsave*.

You can also save the message in a file under another of your directories. If you have a mail message about a particular project or piece of work that you keep in a certain directory, you may want to save that message in the same directory. Let's say you have such a directory, named *project1*, under your login directory. If a mail

message comes in that you want to place in directory *project1*, under a file named *memo*, enter the following response after the question mark:

```
? s project1/memo<CR>
```

If you use the **cd** command to change directories from your login directory to *project1* and then use the **ls** command, you will find that the file *memo* is now listed. (You can use other, more complete path names as well; refer to the "Unix System Capabilities" chapter for instruction on using path names.)

If you want to quit reading messages, enter the following response after the question mark:

```
? q<CR>
```

Any messages that you have left unread are put back in the *mailbox* until the next time you use the **mail** command.

If a long message is being displayed at your terminal, you can interrupt it by pressing the **BREAK** key. This stops the message display, prints the **?**, and waits for your response.

Other responses are available; these are listed in section 1 of the Runtime System manual. The following command recap summarizes what you can expect when using the **mail** command to read messages.

Command Recap

mail - reads messages sent to your login

<i>command</i>	<i>options</i>	<i>arguments</i>
mail	available*	none

Description: **mail** entered by itself displays any messages waiting in the system file *usr/mail* (the mailbox).

Remarks: The question mark (?) at the end of a message indicates that a response is expected. A full list of responses is given in the *UNIX System User Reference Manual*.

* See section 1 of the Runtime System manual for all available options and an explanation of their capabilities.

SENDING AND RECEIVING FILES

In several examples cited so far in this chapter, the need to send files from your UNIX system login to another UNIX system user has come up. Memos, reports, stories, baseball scores--there are numerous items that you can keep in your files. What do you do to send copies of those files to other UNIX system users?

Sending Small Files (*mail*)

The **mail** command uses the redirection symbol < to take its input from a specified file instead of from the keyboard. (For more detailed information on the use of redirection symbols, see *Chapter 7*.) The general format is as follows:

```
mail login < filename<CR>
```

where *login* is the recipient's login name and *filename* is the name of the file containing the information to be sent.

COMMUNICATION TUTORIAL

For example, assume you keep a standard meeting notice in a file named *meetnote*. If you want to send the letter to the owner of login *sarah* using the **mail** command, type the following at the prompt:

```
$ mail sarah < meetnote<CR>
$
```

The system prompt returns to let you know that the contents of *meetnote* have been sent. When *sarah* types in the **mail** command to read her messages, she will receive the standard meeting notice.

Likewise, if you want to send the same file to several users on your system, type in the **mail** command followed by the login names of the users, and then follow these with the < file redirection operator and the file name. It might look like this:

```
$ mail sarah tommy dingo wombat < meetnote<CR>
$
```

The system prompt tells you that the messages have been sent.

If the recipient for your file is on a remote system that can communicate with yours, simply redirect the file with the < operator:

```
mail system!login < filename<CR>
```

For example:

```
$ mail sys10!wombat < meetnote <CR>
$
```

Again, the system prompt notifies you that the message has been queued for sending.

Sending Large Files (*uuto*)

When you need to send large files, you should use the **uuto** command. This command can be used to send files to both local and remote systems. When the files arrive at their destination, the recipient receives a mail message announcing its arrival.

The basic format for the **uuto** command is

```
uuto filename system!login <CR>
```

where *filename* is the name of the file to be sent, *system* is the recipient's system, and *login* is the recipient's login name. The *filename* may be the name of a file or a path name ending in a specific file.

If you send a file to someone on your local system, you may omit the system name and use the following format:

```
uuto filename login <CR>
```

Have You Got Permission?

Before you actually send a file with the **uuto** command, you need to find out whether or not the file is transferable. To do that, you need to check the file's permissions. If they are not correct, you must use the **chmod** command to change them. (Permissions and the **chmod** command are covered in detail in the "Using the File System" chapter of this manual.)

There are two permission criteria that must be met before a file can be transferred using `uuto`:

- The file to be transferred must have read permission (`r`) for *others*, and
- The directory that contains the file must have read (`r`) and execute (`x`) permission for *others*.

This may sound confusing, but an example should clarify the matter.

Assume that you have a file named *chicken*, under a directory named *soup*, that you want to send to another user with the `uuto` command. First you check the permissions on *soup*, which is under your login directory:

```
$ ls -l <CR>
total 35
-rwxr-xr-x 1 reader group1 5598 Mar 313:00 memos
drwxr--r-- 2 reader group1 477 Mar 109:08 lists
drwxr-xr-x 2 reader group1 45 Feb 9 10:43 soup
$
```

Checking the line that contains the information for directory *soup* shows that it has read (`r`) and execute (`x`) permissions in all three groups; no changes have to be made. Now you use the `cd` command to change from your login directory to *soup* and then check the permissions on the file *chicken*:

```
$ ls -l chicken <CR>
total 4
-rw----- 1 reader group1 3101 Mar 1 18:22 chicken
$
```


The output informs you that the file *chicken* has read permission for you, but not for the rest of the system. To add those read permissions, you use the **chmod** command:

```
$ chmod go+r chicken<CR>
```

This adds read permissions to the rest of the system--*group* (**g**) and *others* (**o**)--without changing the previous permissions. Now, checking again with the `ls -l` command reveals the following:

```
$ ls -l chicken<CR>
-rw-r--r-- 1 reader group1 3101 Mar 1 18:22 chicken
$
```

This confirms that the file is now transferable using the **uuto** command. After you send copies of the file, you can reverse the procedure and replace the previous permissions.

Sending a File (*uuto -m, uustat*)

Now that you know how to determine if a file is transferable, let's take an example and see how the whole thing works.

The process of sending a file by **uuto** is referred to as a *job*. When you enter a **uuto** command, your job is not sent immediately. First it is stored in a queue (a waiting line of jobs) and assigned a job number. When the job's number comes up, it is transmitted to the remote system and placed in a public directory there. The recipient is notified by **mail** message and must use the **uupick** command to retrieve the file (this command is discussed later in the chapter).

COMMUNICATION TUTORIAL

For the following discussions, assume this information:

<i>wombat</i>	Your login name.
<i>sys10</i>	Your system name.
<i>marie</i>	Recipient's login name.
<i>sys20</i>	Recipient's system name.
<i>money</i>	File to be sent.

Also assume that the two systems can communicate with each other.

To send the file *money* to login *marie* on system *sys20*, enter the following:

```
$ uuto money sys20!marie<CR>
$
```

The system prompt returns, notifying you that the file has been sent to the job queue. The job is now out of your hands; all you can do is wait for confirmation that the job reached its destination.

How do you know when the job has been sent? The easiest method is to alter the **uuto** command line by adding a **-m** option, like so:

```
$ uuto -m money sys20!marie<CR>
$
```

This option sends a **mail** message back to you when the job has reached the recipient's system. The message may look something like this:

```
$ mail<CR>
From uucp Tue Apr 3 09:45 EST 1984
file /sys10/wombat/money, system sys10
copy succeeded
```

?

If you would rather check from time to time while you are working on the system, you can use the **uustat** command. This command keeps track of all the **uuto** jobs you submit and gives you their status. For example,

```
$ uustat<CR>
1145 wombat sys20 10/05-09:31 10/05-09:33 JOB IS QUEUED
$
```

The elements of this sample status message are as follows:

- 1145 is the job number associated with sending file *money* to *marie* on *sys20*.
- *wombat* is your login name.
- *sys20* is the recipient's system.
- *10/05-09:31* is the date and time the job was queued.

- 10/05-09:33 is the date and time of this particular **uustat** message.
- The final part is the status of the job—in this case indicating that the job has been queued, but has not yet been sent.

If you are interested in just one **uuto** job, you can use the **-j** option and the job number when requesting job status:

```
uustat -jjobnumber<CR>
```

In the example, let's say you enter the **uustat** command with the **-j** option (for job 1145) until you receive the following response:

```
$ uustat -j1145<CR>  
1145 wombat sys20 10/05-09:31 10/05-09:37 COPYFINISHED,JOB DELETED  
$
```

This status message indicates that the job was sent and has been deleted from the job queue—in other words, it has reached the public directory of the recipient's system. There are other status messages and options for the **uustat** command which are described in section 1 of the Runtime System manual.

That is all there is to sending files. You can practice simply by sending another UNIX system user a file. You should practice with a test file until you have the procedures down pat.

The following command recaps give a summary of the **uuto** and **uustat** commands for your convenience.

Command Recap

uuto - sends files to another login

<i>command</i>	<i>options</i>	<i>arguments</i>
uuto	-m and others*	file system!login

Description: **uuto** sends the specified file to the public directory of the specified system. The owner of the login is notified by **mail** that a file has arrived.

Remarks: Files to be sent must have read permission for *others*; the directory above the file must have read and execute permissions for *others*.

The **-m** option notifies you by mail when the file arrives at its destination.

* See section 1 of the Runtime System manual for all available options and an explanation of their capabilities.

Command Recap

uustat - checks job status of a **uuto** job

<i>command</i>	<i>options</i>	<i>arguments</i>
uustat	-j and others*	none

Description: **uustat** checks on the status of all **uuto** jobs sent from your login and displays the results.

Remarks: The **-j** option, followed by a specific job number, displays the status of only the specified job.

* See section 1 of the Runtime System manual for all available options and an explanation of their capabilities.

Receiving Files (*uupick*)

When a file sent by **uuto** shows up in the public directory on your UNIX system, you receive a **mail** message telling you that the file has arrived and where you can find it. To continue our previous example, let's see what the owner of login *marie* receives when she types in the **mail** command, not long after you (login *wombat*) have sent her the file *money*:

```
$ mail
```

```
From uucp Mon May 14 09:22 EST 1984
```

```
/usr/spool/uucppublic/recvie/marie/sys10//money from sys10:wombat arrived
```

```
$
```

The message contains the following pieces of information:

- The first line tells you when the file arrived at its destination.
- The second line up to the two slashes (//) gives you the path name to the part of the public directory where the file has been stored.
- The second line after the two slashes tells you the name of the file and who sent it.

Once you have disposed of the **mail** message, you can use the **uupick** command to store the file where you want it. Type

```
uupick<CR>
```

at the system prompt. The command searches the public directory for any files sent to you. If it finds any, it prompts you with a ? to do something with the file (much like the **mail** command).

Continuing with our previous example, if the owner of login *marie* enters the **uupick** command, she receives the following response:

```
$ uupick<CR>
from system sys10: file money
?
```

After the question mark (?), the command goes to the next line and waits for your response. There are several available responses; we will look at the most common responses and what they do.

The first thing you should do is move the file from the public directory and place it in your login directory so you can see what it is. To do so, type an **m** after the question mark.

```
?
m<CR>
$
```

This response moves the file into your current directory. If you wish to put it in some other directory instead, follow the **m** response with the directory name:

```
?
m directory<CR>
```

If there are other files waiting to be moved, the next one is displayed, followed by the question mark. If not, the prompt returns.

COMMUNICATION TUTORIAL

If you would rather display the next message without doing anything to the current file, press the carriage return key after the question mark.

```
?  
<CR>
```

The current file remains in the public directory until you next use the **uupick** command. If there are no more messages, the system prompt returns.

If you already know that you do not want to save the file, you can delete it by typing in a **d** after the question mark:

```
?  
d<CR>
```

This response deletes the current file from the public directory and displays the next message (if there is one). If there are no additional messages about waiting files, the prompt returns.

Finally, if you want to stop the **uupick** command, type a **q** after the question mark:

```
?  
q<CR>
```


Any unremoved or undeleted files will wait in the public directory until the next time you use the **uupick** command.

Other available responses are listed in the *UNIX System User Reference Manual*. The following command recap summarizes what you can expect from the **uupick** command.

Command Recap

uupick - searches for files sent by **uuto**

<i>command</i>	<i>options</i>	<i>options</i>
uupick	none	none

Description: **uupick** searches the public directory of your system for files sent by **uuto**. If any are found, the command displays information about the file and awaits a response.

Remarks: The question mark (?) at the end of the message indicates that a response is expected. The full list of responses is given in section 1 of the Runtime System Manual.

ADVANCED MESSAGE AND FILE HANDLING (*uucp, mailx*)

Once you master the **mail** and **uuto/uupick** commands, you may decide that you want commands that are more flexible or efficient. If so, you should try the **mailx** and **uucp** commands.

The **uucp** command enables you to send a copy of a file directly to another user's login directory, instead of to the public directory on that user's system. In some cases, you can even copy directly from

files in another login and place the copy in your login directory. The **uucp** command also enables you to rename a file when it reaches its destination.

There are a number of considerations to deal with when using **uucp**, such as file permissions and system security procedures. The **uucp** system is more complex and requires more experience to use than **uuto** and **uupick**.

If you want an electronic mail facility with more features, there is the **mailx** command. This command is an interactive message-handling system that gives you, among other things, the following:

- The ability to use either the **ed** or **vi** text editor for use on incoming *and* outgoing messages,
- A list of waiting messages from which the user can decide which messages to deal with and in what order,
- Several options for saving files, and
- Commands for replying to specific messages and sending copies to other users (both of incoming and outgoing messages).

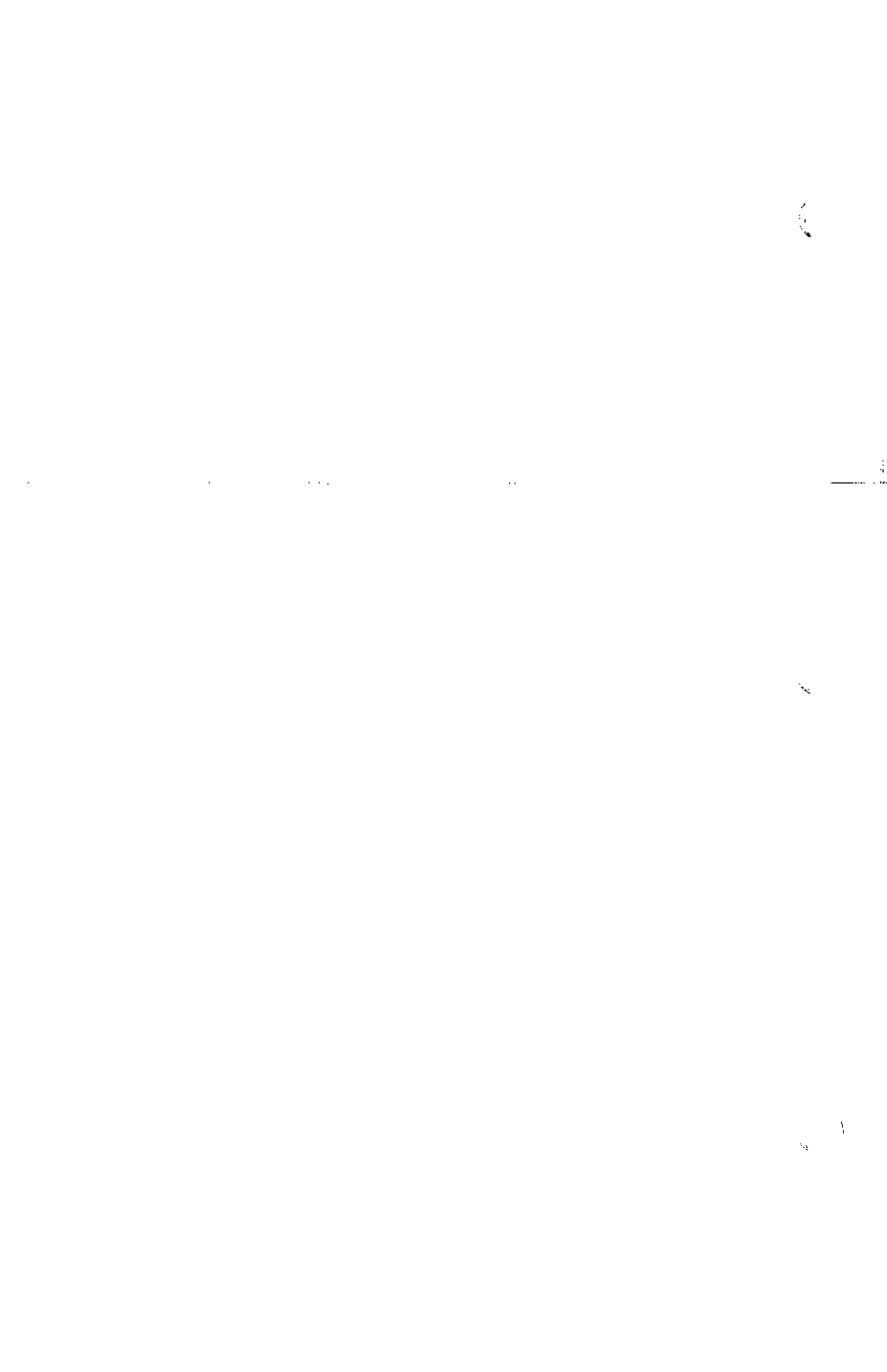
As you might gather, these two commands are complex and are not recommended for the beginning user. Because of this, we do not cover the uses of **uucp** or **mailx** in this guide. However, these commands are mentioned here because they may be available in your UNIX system package and are useful commands to know about.

Once you are thoroughly familiar with the standard tools for user communication, you may want to experiment with the **uucp** and **mailx** commands. Refer to the next chapter for more information on using these commands.

Chapter 12

UUCP ADMINISTRATION

	PAGE
INTRODUCTION	1
PLANNING	1
Extent of the Network	1
Hardware and Line Speeds	2
Maintenance and Administration	2
UUCP SOFTWARE	3
INSTALLATION	3
Object Modules	3
Password File	4
Lines File	5
System File	6
Dialing Prefixes	9
Userfile	9
Forwarding File	1
ADMINISTRATION	12
Cleanup	12
Polling Other Systems	13
Problems	13
DEBUGGING	14



UUCP ADMINISTRATION

INTRODUCTION

This chapter describes how a **uucp** network is set up, the format of control files, and administrative procedures. Administrators should be familiar with the manual pages for each of the **uucp** related commands.

PLANNING

In setting up a network of UNIX systems, there are several considerations that should be taken into account *before* configuring each system on the network. The following parts attempt to outline the most important considerations.

Extent of the Network

Some basic decisions about access to processors in the network must be made before attempting to set up the configuration files. If an administrator has control over only one processor and an existing network is being joined, then the administrator must decide what level of access should be granted to other systems. The other members of the network must make a similar decision for the new system. The UNIX system *password* mechanism is used to grant access to other systems. The file `/usr/lib/uucp/USERFILE` restricts access by other systems to parts of the file system tree, and the file `/usr/lib/uucp/L.sys` on the local processor determines how many other systems on the network can be reached.

When setting up more than one processor, the administrator has control of a larger portion of the network and can make more decisions about the setup. For example, the network can be set up as a private network where only those machines under the direct control of the administrator can access each other. Granting no access to machines outside the network can be done if security is paramount;

however, this is usually impractical. Very limited access can be granted to outside machines by each of the systems on the private network. Alternatively, access to/from the outside world can be confined to only one processor. This is frequently done to minimize the effort in keeping access information (passwords, phone numbers, login sequences, etc.) updated and to minimize the number of security holes for the private network.

Hardware and Line Speeds

There are only two supported means of interconnection by `uucp(1)`,

1. Direct connection using a null modem.
2. Connection over the Direct Distance Dialing (DDD) network.

In choosing hardware, the equipment used by other processors on the network must be considered. For example, if some systems on the network have only 103-type (300-baud) data sets, then communication with them is not possible unless the local system has a 300-baud data set connected to a calling unit. (Most data sets available on systems are 1200-baud.) If hard-wired connections are to be used between systems, then the distance between systems must be considered since a null modem cannot be used when the systems are separated by more than several hundred feet. The limit for communication at 9600-baud is about 800 to 1000 feet. However, the RS232 specification and AT&T Computer Systems Support Groups only allow for less than 50 feet. Limited distance modems must be used beyond 50 feet as noise on the lines becomes a problem.

Maintenance and Administration

There is a minimum amount of maintenance that must be provided on each system to keep the access files updated, to ensure that the network is running properly, and to track down line problems. When more than one system is involved, the job becomes more difficult because there are more files to update and because users are much less patient when failures occur between machines that are under local control.

Lines File

The file `/usr/lib/uucp/L-devices` contains the list of all lines that are directly connected to other systems or are available for calling other systems. The file contains the attributes of the lines and whether the line is a permanent connection or can call via a dialer. The format of the file is

```
type line call-device speed protocol
```

where each field is

- | | |
|--------------------|--|
| <i>type</i> | Two keywords are used to describe whether a line is directly connected to another system (DIR) or uses an automatic calling unit (ACU). An X.25 permanent virtual circuit would use the DIR keyword. |
| <i>line</i> | This is the device name for the line (e.g., <i>ttyab</i> for a direct line, <i>cul0</i> for a line connected to an ACU). |
| <i>call-device</i> | If the ACU keyword is specified, this field contains the device name of the ACU. Otherwise, the field is ignored; however, a placeholder must be used in this field so that the <i>protocol</i> field can be interpreted. |
| <i>speed</i> | The line speed that the connection is to run at. (The speed field is currently ignored if an X.25 link is used.) |
| <i>protocol</i> | This is an optional field that needs only be filled in if the connection is for a protocol other than the default terminal protocol. The X.25 protocol is the only other protocol supported and the single character <i>x</i> is used to select this protocol. |

UUCP

The following entries illustrate various types of connections:

```
DIR ttyab 0 9600
ACU cul0 cua0 1200
DIR x25.s0 0 300 x
```

The first entry is for a hard-wired line running at 9600-baud between two systems. Note that the *acu-device* field is zero. The second entry is for a line with a 1200-baud ACU. The last entry is for an X.25 synchronous direct connection between systems. Note that the *protocol* field is filled in and that the *acu-device* and *line speed* fields are meaningless.

Naming Conventions

It is often useful when naming lines that are directly connected between systems or which are dedicated to calling other systems to choose a naming scheme that conveys the use of the line. In the earlier examples, the name *ttyab* is used for the line that directly connects two systems named *a* and *b*. Similarly, lines associated with calling units are best given names that relate them to the calling unit (note the names *cul0* and *cua0* to specify the line and calling unit, respectively).

System File

Each entry in this file represents a system that can be called by the local **uucp** programs. More than one line may be present for a particular system. In this case, the additional lines represent alternative communication paths that will be tried in sequential order. The fields are described below.

system name Name of the remote system.

time This is a string that indicates the days-of-week and times-of-day when the system should be called (e.g., MoTuTh0800-1730).

The day portion may be a list containing *Su*, *Mo*, *Tu*, *We*, *Th*, *Fr*, *Sa*; or it may be *Wk* for any week-day or *Any* for any day. The time should be a range of times (e.g., 0800-1230). If no time portion is specified, any time of day is assumed to

be allowed for the call. Note that a time range that spans 0000 is permitted; 0800-0600 means all times are allowed other than times between 6 and 8 am. An optional subfield is available to specify the minimum time (minutes) before a retry following a failed attempt. The subfield separator is a ",", (e.g., *Any,9* means call any time but wait at least 9 minutes before retrying the call after a failure has occurred).

device This is either *ACU* or the hard-wired device name to be used for the call. For the hard-wired case, the last part of the special file name is used (e.g., *tty0*).

class This is usually the line speed for the call (e.g., 300).

phone The phone number is made up of an optional alphabetic abbreviation (dialing prefix) and a numeric part. The abbreviation should be one that appears in the *L-dialcodes* file (e.g., *mh1212*, *boston555-1212*). For the hard-wired devices, this field contains the same string as used for the *device* field.

login The login information is given as a series of fields and subfields in the format

[*expect send*] ...

where *expect* is the string expected to be read and *send* is the string to be sent when the *expect* string is received.

The *expect* field may be made up of subfields of the form

expect[-send-expected] ...

where the *send* is sent if the prior *expect* is *not* successfully read and the *expect* following the *send* is the next expected string. (For example, login--login will expect *login*; if it gets it, the program will go on to the next field; if it does not get *login*, it will send *null* followed by a new line, then expect *login* again.) If no characters are initially expected from the remote machine, the string "" (a null string) should be used in the first expect field.

There are two special names available to be sent during the login sequence. The string *EOT* will send an *EOT* character, and the string *BREAK* will try to send a *BREAK* character. (The *BREAK* character is simulated using line speed changes and null characters and may not work on all devices and/or systems.) A number from 1 to 9 may follow the *BREAK* (e.g., *BREAK1*, will send 1 null character instead of the default of 3). Note that *BREAK1* usually works best for 300-/1200-baud lines.

There are several character strings that cause specific actions when they are a part of a string sent during the login sequence.

- \s Send a space character.
- \d Delay one second before sending or reading more characters.
- \c If at the end of a string, suppress the new-line that is normally sent. Ignored otherwise.
- \N Send a null character.

These character strings are useful for making **uucp** communicate via direct lines to data switches.

A typical entry in the *L.sys* file would be

```
sys Any ACU 300 mh7654 login uucp ssword: word
```

The expect algorithm matches all or part of the input string as illustrated in the password field above.

Dialing Prefixes

This file contains the dial-code abbreviations used in the *L.sys* file (e.g., py, mh, boston). The entry format is

```
abb dial-seq
```

where *abb* is the abbreviation and *dial-seq* is the dial sequence to call that location.

The line

```
py 165-
```

would be set up so that entry *py7777* would send 165-7777 to the dial unit.

Userfile

This file contains user accessibility information. It specifies four types of constraints:

1. Files that can be accessed by a normal user of the local machine.
2. Files that can be accessed from a remote computer.
3. Login name used by a particular remote computer.
4. Whether a remote computer should be called back in order to confirm its identity.

Each line in the file has the format

```
login,sys [ c ] pathname [ pathname ] ...
```

where

login is the login name for a user or the remote computer.

sys is the system name for a remote computer.

c is the optional *call-back required* flag.

pathname is a pathname prefix that is acceptable for *sys*.

The constraints are implemented as follows:

1. When the program is obeying a command stored on the local machine, the pathnames allowed are those given on the first line in the *USERFILE* that has the login name of the user who entered the command. If no such line is found, the first line with a *null* login name is used.
2. When the program is responding to a command from a remote machine, the pathnames allowed are those given on the first line in the file that has the system name that matches the remote machine. If no such line is found, the first one with a *null* system name is used.
3. When a remote computer logs in, the login name that it uses *must* appear in the *USERFILE*. There may be several lines with the same login name but one of them must either have the name of the remote system or must contain a *null* system name.
4. If the line matched in (3.) contains a "c", the remote machine is called back before any transactions take place.

The line

```
u,m /usr/xyz
```

allows machine *m* to login with name *u* and request the transfer of files whose names start with */usr/xyz*. The line

```
you, /usr/you
```

allows the ordinary user *you* to issue commands for files whose name starts with */usr/you*. (This type restriction is seldom used.) The lines

```
u,m /usr/xyz /usr/spool
u, /usr/spool
```

allows *any* remote machine to login with name *u*. If its system name is not *m*, it can only ask to transfer files whose names start with */usr/spool*. If it is system *m*, it can send files from paths */usr/xyz* as well as */usr/spool*. The lines

```
root, /
, /usr
```

allow any user to transfer files beginning with */usr* but the user with login *root* can transfer any file. (Note that any file that is to be transferred must be readable by anybody.)

Forwarding File

There are two files that allow restrictions to be placed on the forwarding mechanism. The format of the entries in each file is the same,

```
system
or
system,user,user2,...
```

The file *ORIGFILE* (*/usr/lib/uucp/ORIGFILE*) restricts the access of systems that are attempting to forward through the local system. The file contains the list of systems (and users) for whom the local system is willing to forward. Each entry refers to the system that was the *source* of the original job and not the name of the last system to forward the file. The second file, *FWDFILE* (*/usr/lib/uucp/FWDFILE*), is a list of valid systems that a job can be forwarded to. (It is not necessarily the name of the destination of

a job, but merely the next valid node.) This file will be a subset of the *L.sys* file and can be used to prevent forwarding to systems that are very expensive to reach but to which access by local users is allowed (e.g., links to overseas universities). If neither of these files exist, **uucp** will be perfectly happy to forward for any system. As an example, if the entry for system *australia* were in the *ORIGFILE* but not in the *FWDFILE* on system *mhtsa*, it would mean that system *australia* would be capable of forwarding jobs into the network via system *mhtsa*. However, no systems in the network could forward a job to *australia* via system *mhtsa*.

ADMINISTRATION

The role of the *uucp* administrator depends heavily on the amount of traffic that enters or leaves a system and the quality of the connections that can be made to and from that system. For the average system, only a modest amount of traffic (100 to 200 files per day) pass through the system and little if any intervention with the *uucp* automatic cleanup functions is necessary. Systems that pass large numbers of files (200 to 10,000) may require more attention when problems occur. The following parts describe the routine administrative tasks that must be performed by the administrator or are automatically performed by the *uucp* package. The part on problems describes what are the most frequent problems and how to effectively deal with them.

Cleanup

The biggest problem in a dialup network like *uucp* is dealing with the backlog of jobs that cannot be transmitted to other systems. The following cleanup activities should be routinely performed by shell scripts started from **cron**(1).

Cleanup of Undeliverable Jobs

The **uudemonday** procedure usually contains an invocation of the **uuclean** command to purge any jobs that are older than some fixed time (usually 72 hours). A similar procedure is usually used to purge any *lock* or *status* files. An example invocation of **uuclean**(1M) to remove both job files and old status files every 48 hours is:

```
/usr/lib/uucp/uuclean -pST -pC -n48
```

Cleanup of the Public Area

In order to keep the local file system from overflowing when files are sent to the public area, the **uudemon.day** procedure is usually set up with a **find** command to remove any files that are older than 7 days. This interval may need to be shortened if there is not sufficient space to devote to the public area.

Compaction of Log Files

The files *SYSLOG* and *LOGFILE* that contain logging information are compacted daily (using the **pack** command from the shell script **uudemon.day**) and should be kept for 1 week before being overwritten.

Polling Other Systems

Systems that are passive members of the network must be polled by other systems in order for their files to be sent. This can be arranged by using the **uusub(1)** command as follows:

```
uusub -cmhtsd
```

which will call *mhtsd* when it is invoked.

Problems

The following sections list the most frequent problems that appear on systems that make heavy use of **uucp(1)**.

Out of Space

The file system used to spool incoming or outgoing jobs can run out of space and prevent jobs from being spawned or received from remote systems. The inability to receive jobs is the worse of the two conditions. When file space does become available, the system will be flooded with the backlog of traffic.

Bad ACU and Modems

The ACU and incoming modems occasionally cause problems that make it difficult to contact other systems or to receive files. These problems are usually readily identifiable since *LOGFILE* entries will usually point to the bad line. If a bad line is suspected, it is useful to use the **cu(1)** command to try calling another system using the suspected line.

Administrative Problems

Some **uucp** networks have so many members that it is difficult to keep track of changing passwords, changing phone numbers, or changing logins on remote systems. This can be a very costly problem since ACU's will be tied up calling a system that cannot be reached.

DEBUGGING

In order to verify that a system on the network can be contacted, the **uucico** daemon can be invoked from a user's terminal directly. For example, to verify that *mhtsd* can be contacted, a job would be queued for that system as follows:

```
uucp -r file mhtsd!~/tom
```

The **-r** option forces the job to be queued but does not invoke the daemon to process the job. The **uucico** command can then be invoked directly:

```
/usr/lib/uucp/uucico -r1 -x4 -smhtsd
```

The **-r1** option is necessary to indicate that the daemon is to start up in *master* mode (i.e., it is the calling system). The **-x4** specifies the level of debugging that is to be printed. Higher levels of debugging can be printed (greater than 4) but requires familiarity with the internals of **uucico**. If several jobs are queued for the remote system, it is not possible to force **uucico** to send one particular job first. The contents of *LOGFILE* should also be

monitored for any error indications that it posts. Frequently, problems can be isolated by examining the entries in *LOGFILE* associated with a particular system. The file *ERRLOG* also contains error indications.

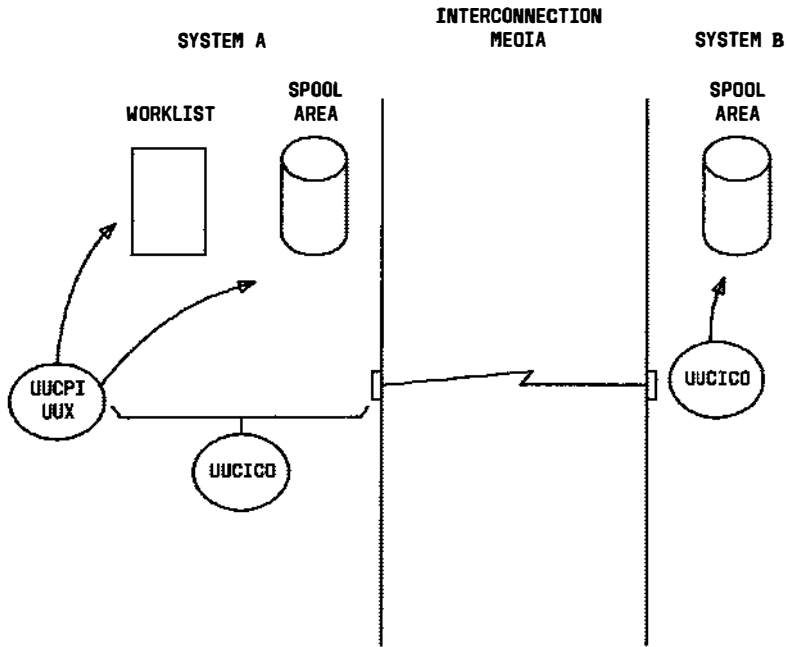


Figure 10-1. Uucp Network Daemon

UUCP SOFTWARE

Figure 10-1 (at the end of this chapter) is an illustration of the daemons used by the **uucp** network to communicate with another system. The **uucp(1)** or **uux(1)** command queues users requests and spawns the **uucico** daemon to call another system. Figure 10-2 (at the end of this chapter) illustrates the structure of **uucico** and the tasks that it performs in communicating with another system. **Uucico** initiates the call to another system and performs the file transfer. On the receiving side, **uucico** is invoked to receive the transfer. Remote execution jobs are actually done by transferring a command file to the remote system and invoking a daemon (**uuxqt**) to execute that command file and return the results.

INSTALLATION

The **uucp(1)** package is delivered as part of the standard UNIX system distribution. It resides in its own subdirectory (called *uucp*) in the commands area and has its own make file (*uucp.mk*). The **uucp** package is installed as part of the normal distribution; however, if it must be reinstalled for any reason, then the sequence

```
make -f uucp.mk install
```

should be executed.

Object Modules

The following object modules are installed as part of the **uucp** make procedure.

1. **Uucp**—The file transfer command.
2. **Uux**—The remote execution command.
3. **Uucico**—The **uucp** network daemon.
4. **Uustat**—Network status command.

UUCP

5. **Uuclean**—Cleanup command.
6. **Uusub**—The command for monitoring and creating a subnetwork.
7. **Uuxqt**—The remote execution daemon.
8. **Uudemon.day**—A shell procedure that is invoked each day to maintain the network. Shell scripts for execution each week (**uudemon.wk**) and each hour (**uudemon.hr**) are also distributed.

Password File

To allow remote systems to call the local system, password entries must be made for any **uucp** logins. For example,

```
nuucp:zaaAA:6:1:UUCP.Admin:/usr/spool/uucppublic:/usr/lib/uucp/uucico
```

Note that the **uucico** daemon is used for the shell, and the spool directory is used as the working directory.

There must also be an entry in the *passwd* file for an **uucp** administrative login. This login is the owner of all the **uucp** object and spooled data files and is usually "uucp". For example, the following is a entry in */etc/passwd* for this administrative login:

```
uucp:zAvLCKp:5:1:UUCP.Admin:/usr/lib/uucp:
```

Note that the standard shell is used instead of **uucico**. If an owner other than "uucp" is chosen, the *make* file for **uucp** (*/usr/src/cmd/uucp/uucp.mk*) must be edited. The line "OWNER=uucp" must be changed to reflect the new owner login.

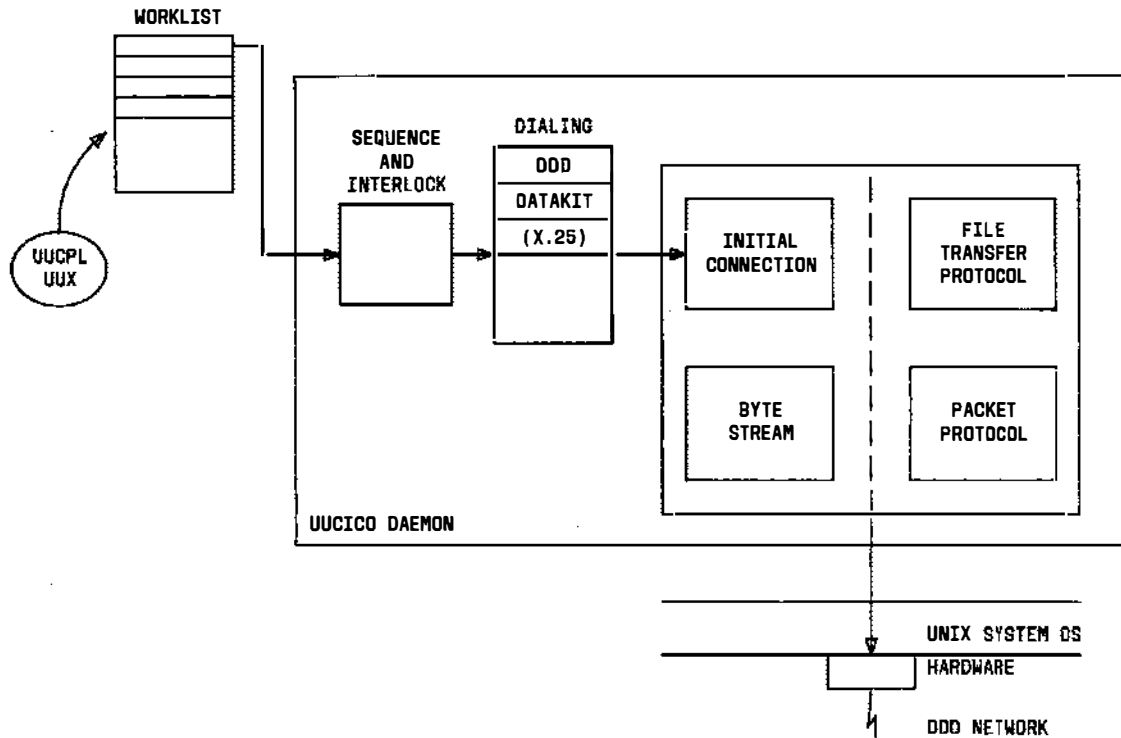


Figure 10-2. Uucico Daemon Functional Blocks



Chapter 13

ADDING DEVICE DRIVERS USING THE LINK KIT

	PAGE
INTRODUCTION	1
What is a UNIX Device Driver?.....	1
THE GENERIC UNIX DRIVER	2
Driver Activities and Responsibilities.....	2
System Buffers.....	3
Data Transfer Between System and User Space.....	3
Sleeping and Waking Processes.....	4
Kernel Timers.....	5
Synchronous and Interrupt Sections of a Driver.....	6
Interrupt Processing.....	6
Critical Sections of the Driver.....	7
How Data Moves Between the Kernel and the Device.....	8
UNIX DRIVER SPECIFICS	9
Types of Devices.....	9
Special Files.....	9
Major and Minor Numbers.....	10
The /dev Directory.....	10
The System and Master Files.....	11
STRUCTURE OF THE DEVICE DRIVER SOURCE FILES	12
Include Files.....	12
General System Data Structures.....	13
Driver Specific Data Structures.....	15
FUNCTION SPECIFICATIONS (Driver Entry Points).....	15
Open.....	15
Close.....	16
Read and Write.....	16
Strategy.....	18
Ioctl.....	19

CODE FOR BRINGING A DEVICE INTO SERVICE.....	20
Interrupt Handler.....	20
Use of Line Disciplines.....	22
Function Naming Conventions.....	22
SYSTEM UTILITY FUNCTIONS.....	22
Sleep and Wakeup.....	22
Setting Processor Priority Levels.....	23
Sleep Priorities.....	26
Timeout.....	28
Allocating Buffer Space.....	29
Buffer Pool.....	29
Clists.....	31
Dynamic Memory Allocation.....	32
DMA Controller Operations.....	34
KERNEL CONFIGURATION	35

ADDING DEVICE DRIVERS TO SYSTEM V/AT

INTRODUCTION

This chapter contains the rules and procedures that should be followed when using the Microport SYSTEM V/AT Link Kit to write device drivers for Microport SYSTEM V/AT UNIX. It is assumed that the reader has user level experience with the UNIX system, some general knowledge of UNIX system concepts, and the capability to write sophisticated C language programs. Writing a device driver carries a heavy responsibility. As part of the UNIX kernel, it is assumed to always take the correct action. Few limits are placed on the driver by the other parts of the kernel, and the driver must be written to never compromise the system's stability.

This chapter contains general information on "generic" UNIX device drivers and describes the scheme for installing drivers in SYSTEM V/AT. Necessary files are obtained via a floppy diskette containing a SYSTEM V/AT Link Kit. When installing a new device driver, users will step through procedures that are consistent with other SYSTEM V/AT software installation and system administration functions.

In addition to the material presented in this chapter, you will find other driver-level information specific to Microport's implementation of UNIX in the files "README" and "doc/link.doc" on the Link Kit diskette.

What is a UNIX Device Driver?

The UNIX operating system can be divided into two parts: one part deals with management of the file system and processes and the second part deals with the management of physical devices such as terminals, disks, tape drives, and network media. To simplify the terminology, this chapter will refer to the first part as the kernel, although strictly speaking, drivers are part of the kernel. The discussion will focus on the second part that contains the driver, sometimes called the I/O subsystem.

Associated with each device is a piece of code, called the device driver, that manages the device hardware. The device driver is responsible for bringing the device into and out of service, setting hardware parameters in the device, transmitting data from the kernel to the device, receiving data from the hardware writing device and passing it back to the kernel, and handling device errors.

One strength of the UNIX system is the ease with which new hardware can be integrated with existing software. The integration process is simple, because the operating system architecture provides a uniform software interface to every device. Processes use the same model when communicating with disks, terminals, printers, or even "pseudo" devices that exist only in software. Every device on a UNIX System looks like a file. In fact, the user-level interface to the device is called a "special file."

The device special files reside under the */dev* directory, and a simple *ls* command will tell you quite a bit about the device. For example, the command *ls -l /dev/lp* might yield the following on SYSTEM V/AT:

```
crw-rw-rw 2 root sys 7, 0 Nov 26 12:33 /dev/lp
```

This says that the "lp" (lineprinter) is a character special device (the first letter of the file mode field is "c") and that major device number 7 and minor device number 0 are assigned to the device. More will be said later about device types, and major and minor numbers.

The Generic UNIX Driver

This section of the chapter addresses issues relevant to drivers on any UNIX system. Throughout this section references are made to how things work on a "generic" or traditional UNIX system, along with some specific details on how the SYSTEM V/AT UNIX system is implemented. The areas of device interrupts and priority levels, in particular, are heavily machine dependent and reflect the SYSTEM V/AT implementation.

UNIX device drivers for different computer systems have many identical characteristics. However, even on the same machine, one driver may be very different from another because of the wide spectrum of functions that drivers perform. Let's first discuss some design issues and examine the common features.

Driver Activities and Responsibilities

A user process runs in a space isolated from critical system data and other programs, protecting the system and other programs from its mistakes. In contrast, a driver executes in the kernel mode, placing few limits on its freedom of action. The driver is simply assumed to be correct and responsible.

This level of responsibility and reliability cannot be avoided. A driver *must* be part of the kernel before it can service interrupts and access device hardware. The existence of the driver is one of the major factors that permits the kernel to present a uniform interface for all devices and to protect processes from some kinds of errors.

The importance of reliable driver code is clear. The driver must not make mistakes that affect any other portion of the system. It should process interrupts efficiently to preserve the scheduler's ability to balance demands on the system. It should use system buffers responsibly to avoid degrading system performance, or requiring that more space be devoted to buffers than is really needed.

This section provides a broad overview of what device drivers do inside the UNIX kernel. The specific details are mentioned later in this chapter. The purpose of the overview is to introduce issues of significance and establish a common language for further discussion. Experienced driver developers will be familiar with much of the information, but those new to UNIX device drivers may find the implications of a multi-tasking environment more complex than expected.

System Buffers

A feature common to most drivers is their use of buffers. There are two types of buffers in a standard System V kernel: clists and system buffers. They differ greatly in size and structure and are meant to fulfill different needs.

System buffers are the size of the largest file system block, 1024 bytes. This buffer pool primarily supports disk I/O operations. The clist manages groups of buffers of much smaller size, typically holding only 64 bytes each. They were created to support I/O typified by lower data rates (i.e., terminal I/O). While drivers may allocate independent buffer pools, this increases the size of the driver, and thus the size of the kernel.

The buffers are a commonly used UNIX resource. The pools are of fixed sizes, though the number of buffers is controlled by constants in the kernel. Whether it uses a private buffer or the public pools, every driver should be written with the finite nature of the machine in mind; space, used for buffering, is taken away from the user processes, so intense buffer use by a driver can reduce the performance of other drivers or require more memory to be devoted to buffers. If more memory must be allocated to buffers, this decreases the memory available for user processes. More information will be provided in the Buffer Pool section on how to obtain and return buffers.

Data Transfer Between System and User Space

The kernel instruction and data spaces are strictly segregated from those of user processes. The need for the kernel to protect itself is obvious. This protection creates the need for a way to transfer information from user space to kernel space and back to user space.

There are several routines for transferring data across the user/system boundary. Some transfer bytes, some words, and others arbitrary size buffers. Each type of operation implies a pair of routines: one for transfers from user space to system space and one for transfers from system space to user space.

At this time, it would be helpful to consider a representative I/O operation and the information transfer across the user/kernel boundary it engenders. As an example, take a request from a process to write a buffer on the disk. The write routine takes the file descriptor, the buffer address in user space, and the length of the data in the buffer as parameters.

The system call causes the processor to transfer from user to kernel mode and to execute the write routine in the generic file interface. When write() realizes that the file is "special" (a device), it uses the appropriate switch table to select the corresponding routine associated with the device. The write routine of the device driver is then faced with a decision.

Since the disk is a shared resource, the device driver may not find it convenient or possible to do the requested write when it is requested. However, when the system call returns, the process assumes that the operation is complete and may do whatever it wishes with its buffer. If the kernel wishes to defer the write to the disk, it must take a copy of the information from user space, keeping it in system space until the write can be done.

Sleeping and Waking Processes

In the previous section, an example of a write operation to the disk introduced several basic concepts. A process might have to wait for the requested information to be read from or written to the disk before continuing. One way that processes can coordinate their actions with events is through the sleep() and wakeup() calls.

Let's consider a read operation in greater detail. When the request is made, the driver has some calculations and setup functions to perform. After these are complete, the request for the information can be made, but there will be a delay before the information is available. The delay, at a minimum, will be due to the retrieval time for the disk. However, it could be much longer if other requests are queued ahead of this one.

Since the UNIX System is a multiuser, multitasking, operating system, it is possible that another job is ready to run and is waiting for the use of the machine. One process should not keep the machine idle while another process is ready to run, so some way must be found to have the first process wait until its information is available. The Sleep/Wakeup mechanism can coordinate this. In the disk access example, the read routine in the disk's driver set would issue a request for the information and put the process to "sleep."

A sleeping process is still considered to be an active process; 'sleeping' is the state or mode of a queue of jobs whose execution is suspended while they wait for a particular event. When the process goes to sleep, it specifies the event that must occur before it may continue its task. This event is represented by a number, typically an address of a structure associated with the transaction. The `sleep()` call records the process number and the event, then places it on the list of sleeping processes. Control of the machine is then transferred to the highest priority runnable process. Data transfer in this period can be done by using a DMA (Direct Memory Access) channel to directly move the data between the device and user memory, if the `physio()` routine is called from the driver read and write routines. This direct transfer is possible as the `physio()` routine locks the process in memory so that it will not be swapped to disk while waiting for the hardware to complete the I/O operation.

When the data transfer is completed, the disk will post an interrupt, causing the interrupt routine in the driver to be activated. The interrupt routine will do whatever is required to properly service the device and issue a `wakeup()` call. It must know what number was used by the process as the sleeping event in order to wake it. This scenario for coordination between asynchronous events appears throughout the kernel.

Kernel Timers

The `timeout()` facility is available for situations that require a sleeping limit for a process. In some cases, a driver must be sure that it is awakened after a maximum period.

This routine takes three arguments: an integer function pointer, a character pointer, and an integer. The integer specifies the period of time in "ticks." Each tick is one-sixtieth of a second in countries where there is sixty-cycle current. The defined constant `HZ*` (see `param.h`) gives the line frequency used by a given kernel. When this period of time has passed, the function pointed to by the first argument to `timeout()` will be called with the second argument as its parameter.

A driver can ensure that it will be able to resume its execution even if no call to `wakeup()` is made by first calling `timeout()` and then `sleep()`. However, this should be done only if truly necessary; as it carries some heavy processing requirements. When the call to `timeout()` is made, it inserts the specified event into the callout table. This data structure is a list of events in a simple array. Insertion of the event requires copying all elements of the list following the inserted event.

* In `SYSTEM V/AT` a patchable variable `hz`, should be used instead of the constant `HZ`. Its value can be examined using the command: `patch system5 hz`, (see `PATCH(1)`).

If the sleeping process is not awakened before the "timeout" event, the specified function will be called. The second argument to the `timeout()` routine could be the event the driver was about to sleep on. When the function is called, it can use this information to call `wakeup()` to wake the driver. The function called from the callout table should also set some internal flag to permit the driver to distinguish between the two ways it can be awakened.

For situations in which the driver must wait for a hardware function to complete, a `delay()` function is provided. This function allows the system to handle other tasks while the driver is delaying. It is not used if an interrupt will be generated when the hardware operation is completed.

Synchronous and Interrupt Sections of a Driver

As described earlier, the system uses system buffers and routines to transfer information across the user or system boundary. Drivers provide the connection between two frames of reference; the process and real-time realms.

The portion of the driver that deals with real-time events is driven by interrupts from devices, and is thus called the interrupt section. The rest of the driver executes only when the process calling the driver is the active process. The execution of this part of the driver is synchronized with the process it serves and will be called the synchronous portion of the driver.

The synchronous portion of the driver, since it has the proper process context, is responsible for organizing the information required for the requested operation. It is responsible for any transfer of information across the user or system boundary. When the request has been properly submitted, the synchronous portion of the driver can do nothing but wait until the requested operation is complete, so it sleeps.

The interrupt driven section of the driver responds to the demands of the device as they come. The synchronous part must leave enough information in common data structures to permit the interrupt routine to figure out what is happening. The interrupt routine is called when an operation is complete. It is responsible for servicing the device, and waking the process waiting on the event. Note that the interrupt routine can be called at any time. It cannot engage in any activity that depends on process context.

Interrupt Processing

The previous section defined the interrupt and synchronous portions of a driver and mentioned that the interrupt portion is driven by real-time events. The events are demands for attention from the controlled devices.

When a device requests some software service, it generates an *interrupt*. Each device can interrupt the system at a specific *priority level*. If the currently executing code has not blocked interrupts at that level, it will immediately save its status and *trap* to an interrupt handler. The interrupt routine in the driver must determine the cause of the interrupt and take appropriate action. If the synchronous portion of the driver was waiting for this event, the interrupt routine should issue a call to `wakeup()`.

Critical Sections of the Driver

So far, the discussion has been centered around a particular interrupt occurring in isolation. Though helpful, this view is unrealistic and potentially misleading. Interrupts from all the devices on the system can occur at any time, and the implications of this are important. The relationship between the synchronous and interrupt portions of the driver is affected, as are those between drivers sharing data.

When two sections of kernel code have a common interest in specific data, they must be careful to coordinate their efforts. If an interrupt switches control of the system to the interrupt driven portion of the driver, then manipulation of the common data may be caught in the midst of its work. This could render the information invalid and inconsistent.

These concerns are grouped under the general heading *critical sections*. The importance of the issue is clear; the integrity and accuracy of the data used by drivers is at stake. The word sections refers to the portions of code that manipulate the common data, rather than the data itself. Thus, a *critical section* of code is one that manipulates data that is of concern to another piece of code capable of interrupting the first.

A routine in the kernel that has a critical section must have a way to protect itself from interruption when manipulating critical data. A set of subroutines that permit code to Set the Priority Level (spl) of the processor solves the problem, and are listed under "Setting Processor Priority Levels" in the "System Utility Functions" section of this chapter. A clear understanding of the need for these routines can be achieved by examining a detailed scenario.

Imagine a section of code in the synchronous portion of a driver that manipulates status flags. Such flags are frequently used to communicate between the synchronous and interrupt portions of a driver. Consider also, that the interrupt portion has code that manipulates those flags. Finally, realize that the manipulations do not take place in a single machine operation.

Consider what happens if the synchronous portion of the driver receives a request that requires it to manipulate the values of several flags, but in the midst of the

manipulation the device gives an interrupt, transferring control to the interrupt portion of the driver. The interrupt routine decides that it must consult the flag values to make some decision and then set them to new values.

The flags are in a *positively* incorrect state, because the synchronous routine has only partially finished changing the flags when the interrupt routine took over. This may cause the interrupt routine to take appropriate action, or it may make a harmless, but incorrect decision. Assume that the interrupt routine does not run amok, but simply looks at the flags, makes decisions, and changes a couple of flag values. Then when the interrupt returns, the synchronous portion of the code, unaware that it was interrupted, finishes the changes it had started.

The section of code in the synchronous routine that manipulated data of interest to the interrupt routine is the *critical section*. Whether the data manipulated in a *critical section* is changed by the interrupting routine is unimportant. The fact that the interrupting routine uses the *critical section* is sufficient. It proves that any portion of code that can be interrupted and that manipulates data of interest to the interrupting code, is a *critical section*. When a critical section is identified, it can be protected from interruption by a call to an *spl* routine of the proper level.

How Data Moves Between the Kernel and the Device

The previous discussions assume that the data moves magically between the memory accessible to the kernel and the device itself. This is a machine dependent detail, but it is instructive to examine how this is done. Some machines require the Central Processing Unit (CPU) to execute special I/O instructions to move data between a device register and the addressable memory or to set up a block transfer between the I/O device and memory (this process is often called DMA - Direct Memory Access). Another scheme, known as memory mapped I/O, implements the device interface as one or more locations in the memory address space.

In either case, the operating system usually provides function calls that let drivers access the data in a general way. The SYSTEM V/AT UNIX System implementation provides `inb()` to read a single byte from an I/O address and `outb()` to write a single byte. The syntax of these function calls is shown below:

```

    inb(addr)
    int addr;

and

    outb(addr, data)
    int addr;
    char data;
```


As described earlier, it is the driver's job to copy this data between the kernel address space and the user program's address space whenever the user makes a `read()` or `write()` system call.

The SYSTEM V/AT implementation also supports DMA controller functions for certain devices wired to a DMA channel which can transfer a block of data at a time. The DMA controller has control registers defining DMA start address and word (or byte) count that the driver must manipulate. For more detail on DMA operations see the section on "DMA Controller Operations" later in this chapter.

UNIX Driver Specifics

Types of Devices

There are two classes of devices: block and character. Block devices are addressable. As the term implies, the data on the device is formatted and addressed in "blocks." The term "character device" is a misnomer that should be "raw device," implying that the data being read is raw or unformatted; the device drivers and user programs assign semantics to the data, not the UNIX file system. A physical device could be represented as both a block and a character device in a system configuration, implying that the system can access the device in two ways.

Although some device drivers are normally associated with hardware devices, some drivers may have no hardware counterpart. These devices are often referred to as pseudo devices. For example, a trace driver may log certain classes of events. User programs write to the driver to record the events and read from the driver to recall the information. The trace driver would *malloc* a large buffer for storing the data. No hardware is associated with the driver, and the driver interfaces with software only.

Special Files

The UNIX system treats a device as if it were a file. That is, when a user program wishes to access a device, it accesses the file that is associated with that device. These special files are sometimes called nodes or device nodes. The system calls that access regular UNIX system files such as *etc/motd*, or *etc/passwd* are, therefore, the same calls that access device (such as *ldev/console*). The system calls are `open()`, `close()`, `read()`, `write()`, and `ioctl()`. See the "Function Specifications (Driver Entry Points)" section in this chapter that describes the system calls at the driver level in detail. Note that devices which are represented as both a block device and a character (raw) device have two special files associated with them, one of each type.

Major and Minor Numbers

The device major numbers are used by the system to determine which device driver to execute when a user reads or writes to or from the special file. The system maintains two tables for mapping I/O requests to the drivers. There is one table for "character special," and a second table for "block special." This implies that there are two sets of major numbers: one for character devices and the second for block devices. Both start at zero and are numbered up to the last used major number (with an upper limit of 32 for SYSTEM V/AT). If you do an *ls -lR /dev*, you may find that two very different devices have the same major number. That's probably because one is a "block special," using the block major number, and the other is "character special," using the character major number. For those drivers that are both block and character devices, (for example, the floppy driver), one major number of each type must be assigned. In this case, the actual numbers may be different and generally are different (the floppy disk may be assigned block major number 1 and character major number 6).

The minor number is entirely under control of the driver writer and usually refers to "subdevices" of the device. These subdevices may really be separate units attached to a controller. For example, a disk device driver may talk to a hardware controller (the device) to which several disk drives (subdevices) may be attached. The UNIX system accesses different subdevices using the different minor numbers.

Major numbers are assigned by the driver writer or the system administrator. The *mknod* command is used to create the files (or nodes) to be associated with the device. The device is configured in the kernel by modifying the System and Master files described below.

The /dev Directory

The Device file may exist anywhere in the UNIX file system, but by convention, all device files are contained under the directory */dev*. The names of the files are generally derived from the names of the hardware; a convention that allows users to know what the device is by looking at the filename. It would be confusing if the file */dev/tty* were a disk. Part of the name of the Device file usually corresponds to the unit number of the device to be accessed via the file, or specifically, the minor number.

A new convention of SYSTEM V/AT and other UNIX systems is that */dev* can contain subdirectories that hold the nodes for all the subdevices of a particular type. This reduces the clutter in the */dev* directory. For example, */dev/dsk* contains all the "block special" files for the floppy and hard disks, and */dev/rdisk* contains all the "character special" files.

The Device file may exist in the file system even though the device is not configured in the running system. If a user attempts to access the device, or more specifically, the special file, an error will result on the system call. Conversely, the device may be configured into the running operating system without the Device file in the file system, in which case the device is inaccessible.

The System and Master Files

Associated with device drivers are two device configuration files: the Master file and the System Device file (also known as the *dfile*). See Section 1 for manual page on CONFIG(1M) and Section 4 of the Software Development manual on MASTER(4) for a description of the System Device file and Master file formats.

The Master file contains the device name (8 characters or less), definition of what functions the device supports (field 3 has an "r" if the *read* function is implemented, w if *write* is implemented, etc.), definition of block and/or character major number, and other descriptive information about the driver.

The System Device file contains information on how the device is installed in the system. That is, the number of units (subdevices), *interrupt vector number (IVN) used*, and other local information. In SYSTEM V/AT, if you add a driver to the system you must create a one-line entry for the driver in each of the Master and System Device files.

In the Master file the driver is defined by a line containing nine fields separated by tabs. For instance, to add a driver for a cartridge tape to Microport SYSTEM V/AT, we might create the following Master file line:

```
tape 47 ocrwi c ct 0 9 2
```

where

tape is the driver name

47 is the Interrupt Vector Number (IVN) assigned to the cartridge tape device.

ocrwi are the handlers provided by the driver: open, close, read, write and ioctl.

c defines the driver as a raw (character) type device.

ct is the driver prefix used in driver functions and data structures to differentiate them from names used by other device drivers.

0 is the block major device number. In this case, there is no block tape device, so this field is 0.

9 is the character major device number.

2 indicates there can be a maximum of two tape units controlled by this driver.

In the System Device file, the tape driver requires the following line:

```
tape 0 2
```

where

- tape* is the driver name as defined in the Master file.
- 0 is the Interrupt Vector Number, usually 0 to indicate that the Master file IVN should be used.
- 2 is the number of tape units attached to the tape controller, or 0 to indicate that the value should be the maximum number of units defined in the Master file.

The kernel configuration procedure is described in the section "Kernel Configuration," later in this chapter.

Structure of the Device Driver Source Files

Include Files

Every file in the operating system source code includes header files containing declarations of global data structures. De facto coding standards prohibit nesting one include file inside another. The source code for device drivers need not be contained in a single file, and indeed, programmers should subdivide the driver among several files if it is large. Even if the driver is contained in a single file, programmers should follow convention and declare the driver data structures in new, driver-specific header (*.h*) files. The definition of the data structures (the place in the source code where the compiler allocated memory storage) should be of the form *extern* in a *.c* file, usually the driver source file.

Sometimes data structures defined in the driver are configuration dependent. That is, if the driver needs to allocate storage for each subdevice, a method is needed to allocate based on the number configured. For SYSTEM V/AT, both a *#define* and a global variable for the driver unit count are created at configuration time. The *#define* is made available to the driver source file by including the file *cficonfig.h*. Each driver defined in the System Device file has a *#define* in *cficonfig.h* that consists of the prefix as defined by the first field in the Master file entry followed by "_0". For instance, for the cartridge tape driver defined above, the *cficonfig.h* file contains the line:

```
#define TAPE_0 2
```

If the driver contains a 512 byte buffer for reading and writing tape blocks, it is necessary to create a different buffer for each tape unit, so that both units can be used simultaneously without interfering with each other. The two buffers would be defined in the Driver Source file with the following code segment:

```
#include cf/config.h.

unsigned char ct_buf [TAPE_0];
```

In this example TAPE_0 is defined as 2.

The configuration process also defines a global variable consisting of the driver prefix followed by *_cnt*. For the tape driver, the following external declaration could be placed in the driver source files:

```
extern int ct_cnt;
```

With the *master* and *dfile* lines given above, *ct_cnt* is initialized to 2.

Driver filenames conventionally contain the device name as part of their names. Assume the streaming cartridge tape driver consists of two *.c* files, *ct.c* and *ctint.c*, and one header file, *ct.h*. The names suggest that the files are associated with the new *tape* device, and that the *ctint.c* file contains a protocol for the device. The header file, *ct.h*, may contain a declaration such as the following example:

```
struct ct_state {

    char ct_flags;
    int ct_port;
    int ct_chan;
    char ct_status [ct_stat_512];

};
```

and the *.c* files should contain the line

```
#include "sys/ct.h"
```

General System Data Structures

Driver programmers must not change standard system header files such as the *proc* file, the *user* file, or the *inode* file. Since the drivers are a separate part of the system, it is unacceptable to introduce new data structures and new "hooks" into standard system data structures to accommodate a private driver. In addition, changing system data structures could cause user level programs to work incorrectly if they rely on the system data structure. For example, changes to the process table usually require recompilation of the *ps* command. Driver programmers should likewise refrain from tampering with kernel source files.

Unfortunately, driver source code must contain some standard "include" files to allow the driver access to system utilities and data structures commonly used to return information to the kernel. These include files are provided in the `sys` directory on the Link Kit diskette or in the `usr/include/sys` directory. The list below defines a few of the more commonly used include files.

- *sys/types.h* - Basic system data types
- *sys/param.h* - Fundamental system parameters
- *sys/dir.h* - Directory structure definition
- *sys/signal.h* - Definition of system signals
Used if the driver sends signals to user processes.
- *sys/user.h* - The user structure definition

The driver must include *dir.h* and *user.h* if the error field `u.u_error` is set in the driver or if the fields `u.u_base` or `u.u_count` are used (see the "Read and Write" section in this chapter). The error field gives error information to the kernel, and the information later returns to the user program. If the driver includes *user.h*, it must first include *signal.h* and *dir.h* because of interdependencies between the three header files.

- *sys/conf.h* - Definition of device switch tables
Needed if the driver uses line disciplines (see "Use of Line Disciplines" section in this chapter).
- *sys/file.h* - Definition of file structure
Needed if the driver uses control flags such as "no delay" (FNDELAY).
- *sys/buf.h* - Definition of the `buf` (system buffer) structure
Needed if the driver uses the system buffer pool (see the "Buffer Pool" section in this chapter), or will initiate DMA operations directly to and from user memory.
- *sys/iobuf.h* - Definition of the driver state structure
This is needed if the driver uses the system buffer pool or initiates DMA operations. It maintains the driver state and active I/O buffer.
- *sys/tty.h* - Definition of the `clist` structure
Needed if the driver uses clists (see the "clists" section in this chapter).
- *sys/sysmacros.h* - Useful macros for units conversion, extraction of major and minor device numbers, etc.
- *sys/errno.h* - System wide error numbers
In the event a driver returns an error, it sets `u.u_error` field of the user structure (see *user.h*) to one of these errors.

- *sys/8259.h* - Defines the default interrupt masks for each processor interrupt priority level (see section "Setting Processor Priority Levels").
- *sys/8237.h* - Defines used when the driver initiates DMA operations (see section on "DMA Controller Operations").
- *sys/file.h* - Defines I/O block flags passed to the driver open and close functions.
- *sys/ioctl.h* - The ioctl union definition used by all device driver ioctl routines.

Driver Specific Data Structures

Naming Conventions

The names of driver data structures and variables should have the driver name in the prefix to ease program readability and debugging and to avoid conflict with other variables in the system with the same name. For example, in the case of the cartridge tape driver containing the data structure *ct_state*, the prefix *ct_* identifies them as belonging to the cartridge tape driver.

Unit Numbers

As previously mentioned, drivers frequently "drive" several hardware units, as a terminal driver may "drive" many terminals. Each terminal has a unit number corresponding to the minor number of the device file. Drivers typically contain a data structure that contains a flag field to record the device status, such as open, sleeping, waiting for data to drain, etc. Except for the inclusion of a flag field, the contents of the data structure are device dependent. However, there should be one entry per unit defined in the driver source file and declared in the header file. A sample declaration of the data structure for the cartridge tape device, *ct*, was defined previously. Each *ct* device should have one of these data structures. For devices using the system buffer pool this data structure can be the *jobuf* structure defined in *jobuf.h*.

Function Specifications (Driver Entry Points)

This section describes the functions that form the driver interface to the kernel. For a raw device, they are open, close, read, write, and ioctl. For a block device, they are open, close, and strategy. A driver need not contain every routine if it is irrelevant (a lineprinter driver usually does not have a read routine). If a device is represented as both raw and block, the driver must contain all the functions as appropriate.

Open

The kernel calls the driver open function as a result of an open system call for the device file.

```
ctopen(dev,flag)
int dev, flag;
```

The parameters of the driver open function are the minor device number of the device `file` and the flags supplied in the "oflag" field of the open system call, corresponding to flag values in the header file "file.h." The minor device number usually corresponds to the unit number of the physical device being opened. The responsibility of the open routine is to establish a "connection" between the user and the device. The first time the open function is called the driver might need to initialize the device. If multiple opens do not make sense, it is the responsibility of the driver open function to return an error by setting the `uu_error` field:

```
uu_error = ENXIO;
```

Close

The kernel calls the driver close function with the minor number of the device `file` as its parameter.

```
ctclose(dev)
int dev;
```

The responsibility of the close function is to end the connection between the user process and the previously opened device and to "clean up" the device (hardware and software) so that it is ready to be opened again.

Read and Write

The kernel calls the driver read and write routines to read (write) data from (to) the device specified by the unit number, the only parameter.

```
ctread(dev)
int dev;
and
ctwrite(dev)
int dev;
```

Drivers for "raw" devices contain these routines. Because user programs and the operating system execute in different address spaces, the I/O cannot take place directly from the device to the user program (unless the device is also a "block" device, as will be explained at the end of this section). There must be a system buffer between them. When reading, the driver must receive the data from the device in a read buffer, and then copy the data from the buffer to the local buffer of the user process.

When writing, the driver must copy the data from the local buffer of the user process, and then transmit the data from the buffer to the device. The buffers can be a private driver structure or one obtained by use of the system utility routines described in the "Allocating Buffer Space" section in this chapter.

In the driver read routine, the system variable *u.u_base* is the address of the buffer in the user program address space, and the variable *u.u_count* is the number of bytes remaining to be read.

The functions *copyout()* or *subyte()* should be used to copy the data from the driver buffer to the user's local buffer:

```
copyout(ptr_to_driver_buffer, u.u_base, n)
subyte(u.u_base, char_c)
```

where *n* is the number of bytes the function copies from the buffer (pointer *ptr_to_driver_buffer* in the *copyout()* function, and *char_c* is the single character that *subyte()* copies to the user buffer. The driver should use *copyout()* for copying more than one byte of data, and it should use *subyte()* for copying one byte of data. After the function calls, the driver should increment the value of *u.u_base* by *n* and decrement the value of *u.u_count* by *n* (the number of bytes transferred). (For *subyte()*, the value of *n* is 1.) If either function returns a non-zero value, then *u.u_error* should be set to *EFAULT* to indicate the error.

The driver write routine is similar to the read routine, except that the routines *copyin* and *subyte* are used to copy data from the user buffer to a system buffer.

```
copyin(u.u_base, ptr_to_driver_buffer, n);
subyte(u.u_base);
```

The system is not responsible for "bad" addresses set in a *u.u_base* (set as a result of the user system call). If the user is reading in 512 bytes from the device into a user data structure that is 256 bytes long, it is not the system's job to detect the error. The user program must make sure that the data returning from a read system call will not overflow the user buffer.

The read and write routines are responsible for resetting the device hardware so that later calls work correctly. They are also responsible for "cleaning up" errors, and keep appropriate statistics.

Raw devices that are also block devices can avoid copying data into an intermediate buffer by using the *physio()* routine and the *strategy()* routine.

```
physio(strategy, bp, dev, rdwr_flag);
int (*strategy)();
struct buf *bp;
int dev;
int rdwr_flag;
```

The *strategy* parameter is the name of the *strategy()* routine for the device, described in the next section. The buffer header pointer, *bp*, is a locally allocated buffer header, not one allocated from the buffer pool (see the "Buffer Pool" section) because the *physio()* routine assigns the data pointer, in the buffer, to

the location in the user program (*u.u_base*) where the data transfer should come from or go to. If the *bp* parameter is null, the *physio()* routine assigns a buffer internally (probably a safer way to invoke it). The *dev* parameter is the device number, and the *rdwr_flag* should be *B_READ* or *B_WRITE*, appropriately. The *physio()* routine will call the device strategy routine internally and set up the data transfer directly between the device and user space. This can only be done if the device is a block device as well as a raw device.

The *copyin* and *copyout* functions should not be used by the interrupt handler, rather they should be used in the Read and Write routines after the process wakes up, that is, after *physio()* returns. The interrupt handler should just wake up the process by calling *iodone()*.

Strategy

Drivers of block devices must define a strategy routine to start I/O to and from the devices.

```
ctstrategy(bp)
struct buf *bp;
```

The kernel calls the strategy routine with a parameter that is a pointer to a buffer header containing all the information about the I/O operation. For example, the definition of a strategy function for the disk driver called *dsk* would look like the following:

```
#include "sys/buf.h"

dskstrategy(bp)
struct buf *bp;
{
/* body of strategy routine */
}
```

The strategy routine uses the following fields in the buffer, but it should not set them.

- *dev_t* *b_dev*;

b_dev contains the major and minor number of the device where the I/O is to occur. The minor number is contained in the 8 low order bits, and the major number is contained in the 5 next low order bits. The 3 high order bits should not be used.

- *daddr_t* *b_blkno*;

b_blkno is the block number on the device where the I/O is to occur.

- unsigned int b_bcount;
b_bcount is the number of bytes to be transferred by the I/O operation.
- caddr_t b_un.b_addr
b_un.b_addr is the address of the data in the buffer. The data array is SBUFSIZE bytes long.
- int b_flags
b_flags gives the buffer status. If the B_READ bit is set, then the I/O operation is to read from the device; if the B_WRITE bit is set, then the I/O operation is to write the device.

If the strategy routine finds an error in setting up the I/O or if the device reports an error via an interrupt, the driver should set the following fields.

- int b_flags
b_flags should have the B_ERROR bit or'ed in. The driver should not assign a value to b_flags because that may erase other bit patterns that the kernel relies on.
- char b_error
b_error should be set to an appropriate error value. Typical values are EIO for some physical I/O error, ENXIO for attempting I/O on bad device or bad device address, or EACCES for attempting to access a device illegally. The kernel later sets u.u_error with the value of b_error so any appropriate value for u.u_error could be set.
- unsigned int b_resid;
b_resid should be set to the number of bytes that have not been transmitted.

ioctl

The driver ioctl routine controls hardware parameters and the interpretation of data as it passes through the driver via read and write.

```
ctioctl(dev, cmd, arg, mode)
int dev, cmd, arg, mode;
```

The routine takes 4 parameters:

- dev - the minor device (unit) number
- cmd - A command argument from which the driver ioctl function interprets the type of operation the driver should perform. The command types vary across the range of devices. Drivers that use an ioctl function typically

have a command to "read" the current ioctl settings and at least one other that sets new settings. The kernel does not interpret the command type; so a driver is free to define its own commands.

By convention, ioctl commands are set to complex numbers to help guard against accidental misuse by users. A common technique is to pick the first letter of the device name and left shift the ASCII code by 8, then "or" in a command code into the lower 8 bits. The values for ioctl commands are usually defined in the driver header file so that both the driver and user programs can access the commands via *#defines*.

- **arg** - An arbitrary argument that can pass parameters between a user program and the driver. The argument can be the address of a structure in the user program that contains settings for the driver or hardware. The driver reads the settings from the user program via the `copyin()` function and does the appropriate operations. Similarly, the driver collects current settings and uses the `copyout()` function to write the data into the user program structure. Alternatively, the argument may be an arbitrary integer that has some meaning to the driver. The interpretation of the argument is driver dependent and usually depends on the command type; the kernel does not interpret the argument.
- **mode** - An argument (need not be used) that contains values set when the device was opened. The driver can use the mode to determine if the device unit was opened for reading or writing, if necessary, by checking the `FREAD` or `FWRITE` setting.

Code for Bringing a Device into Service

There may be requirements for writing an initialization code to bring a device into service. Often this can be done in the open routine by creating a flag to determine if this is the first open (generally after a reboot of the system) of the device. If such code is necessary, the driver should contain an initialization function which follows the naming convention previously described. For our example, the initialization function would be called `cinit()`.

If the initialization routine must be called at system initialization time, the Master file entry for the driver should contain an "s" in field 3, in addition to the other driver functions that are supported.

Interrupt Handler

As previously described, hardware interrupts cause the processor to stop its current execution stream and to start executing an instruction stream that services the interrupt. The system identifies the device causing the interrupt and accesses a table of interrupt vectors to transfer control to the interrupt handler for the device.

The exact mechanism of associating interrupt vectors with interrupt handlers varies on different UNIX systems. The discussion here assumes the system finds the correct interrupt routines on receipt of the device interrupt, and it assumes that the system executes the interrupt routine at a processor execution level high enough to prevent more interrupts of that type.

It is important to note that the device driver `open`, `close`, `read`, `write` and `ioctl` functions are called as the result of the user process performing a system call. This means that they are executed only when the currently running process is the user process, and the *u area* fields correspond to the calling process. In contrast, when the device driver interrupt routine executes, the currently running process is probably not the user process (it is probably sleeping, waiting to be awakened in I/O completion). Thus the interrupt handler should only set flags and then awaken the user process by calling `iodone()`. `iodone()` will cause the user process to become ready to run after the interrupt handler returns.

For the SYSTEM V/AT, there are a limited number of available interrupts. For more information on this and other machine dependent aspects of SYSTEM V/AT interrupt architecture, see the "Setting Processor Priority Levels" section in this chapter.

The device interrupt handler routines handle device interrupts, which are the device responses to data transfers and requests. System software cannot predict when a device will interrupt the system. Typically, a system call blocks, or *sleeps*, on an event, awaiting the device to interrupt. The device interrupt causes the system to invoke the interrupt handler that in turn awakens the blocked system call. For instance, device `open` routines may block until the device interrupts and "announces" its connection. Or device `read` routines may block until the device interrupts and "announces" that data has arrived and can be read into the system.

Upon receipt of the interrupt, the kernel evokes the driver interrupt handler.

```
ctintr(dev)
int dev;
```

where *dev* indicates the subdevice associated with the interrupt. The interrupt handler must identify the reason for the interrupt (device connect, write acknowledge, data available) and set or clear device state bits as appropriate. It can also awaken processes that are sleeping (see the "Sleep and Wakeup" section in this chapter), waiting for the event corresponding to the interrupt. Interrupt handlers must not set any fields in the *u area*, particularly *uu_error*, because the interrupted process is independent from the interrupt. For the same reason, interrupt handlers must not call `sleep()`, `delay()`, `subyte()`, `fubyte()`, `copyin()` or `copyout()`.

Use of Line Disciplines

Line disciplines are modules that interact with a driver to massage the data as it passes between the kernel and the device driver. The driver controls the hardware, but software manipulation of the data takes place in the line discipline module. It would be natural to think of networking drivers such that the networking driver controls the hardware medium, and the network protocol is contained in a line discipline. But historically, line disciplines have been associated mostly with terminals. Although used by the SYSTEM V/AT console and drivers, detailed discussion of line disciplines is beyond the scope of this chapter.

Function Naming Conventions

The names of the driver `open`, `close`, `read`, `write`, `ioctl`, `strategy`, `init`, and `interrupt` routines must be prefaced by the generic driver name. For example, the names of the routines for the tape driver are `ctopen()`, `ctread()`, `ctwrite()`, `ctioctl()`, and `ctintr()`. There are no restrictions on names for other functions in the driver, but it is best to preface the function names with the driver name for identification purposes. This will help avoid defining a function already defined in other parts of the operating system.

System Utility Functions

The driver calls kernel routines to perform system level functions, many of which were introduced in the "Driver Activities and Responsibilities" section in this chapter. The following paragraphs describe the syntax and the use of these kernel functions.

Sleep and Wakeup

As described in the "Sleeping and Waking Processes" section in this chapter, drivers must sometimes suspend or block their execution to await certain events, where an event is a system state in hardware or software. The driver waits by calling the `sleep` function, and the system does a context switch and schedules another process.

The `sleep` function takes two parameters: the address (signifying an event) upon which the process will sleep, and a priority value that is assigned to the process when it is awakened.

```
sleep(addr,pri)
caddr_t addr;
int pri;
```

The address used for sleeping is an arbitrary address that has no meaning except to the corresponding `wakeup()` function call. The sleep addresses are usually taken from the entry in the device data structure of the device that the process is accessing to guarantee uniqueness across the system. When a process goes to sleep awaiting an event, the driver should set a flag in the device data structure indicating the reason to sleep.

```
driver.state |= condition;
sleep(&driver.state, PRIORITY);
```

Someone, either an interrupt handler or another process, will later call the `wakeup()` function to awake the sleeping process. The code invoking the `wakeup()` function should check for a particular flag bit, indicating the reason that the process is sleeping. The driver then calls `wakeup()` with one parameter, namely the address where a process could be sleeping.

```
wakeup(addr)
caddr_t addr;
```

It is best for code readability and for efficiency to have a one-to-one correspondence between events and sleep addresses: one address should not be used for sleeping for two events. Again for clarity, there should be one bit in the flag field corresponding to every sleep event, and hence, to every sleep address. The `wakeup()` function awakes all processes sleeping on the address, enabling them to execute when the scheduler chooses them after the interrupt routine exits. If no process is sleeping on the address when `wakeup()` is called, `wakeup()` returns with no bad side effects.

It is illegal to call `sleep` when handling an interrupt, since a process independent of the device could have been executing when the device interrupted. If the interrupt handler goes to sleep, the process that was interrupted is effectively put to sleep for reasons beyond its control. Additionally, and far more important, sleeping in an interrupt handler could cause the system to crash in some UNIX system implementations because of the interdependency of the process context switch mechanism and interrupt levels. The interrupt handler must, therefore, not invoke other functions that could lead to a call to `sleep()`.

Setting Processor Priority Levels

As described in the "Critical Sections of the Driver" section in this chapter, the system allows devices to interrupt the processor and handles the interrupts immediately. The integrity of system data structures could be destroyed if an interrupt handler were to manipulate the same data structures as a process executing in the driver.

To prevent such problems, the system has special functions that set the processor execution priority level (`spl`) to inhibit interrupts below a specific level.

Without use of the `spl` function, the process could check the condition bit, find it true, and attempt to call `sleep`. But if an interrupt occurred before the process called `sleep`, and the interrupt handler checked the condition bit to determine if a process was sleeping, it would assume the process was asleep and call `wakeup` to awake it. Consider the following code:

```

if (driver.state & condition)
{
    driver.state &= ~condition;
    wakeup(&driver.state);
}
    
```

By the time the interrupted process calls `sleep()`, it will have missed the `wakeup()` call, and another `wakeup` call may never come. By bracketing the calls to `sleep()` with `spl()` function calls, the driver prevents the race condition.

```

spl5();
driver.state |= condition;
while (driver.state & condition)
    sleep(&driver.state, PRIORITY);
spl0();
    
```

The function `splbio()` should be used for block devices doing physical I/O, such as disk and tape drives. In System V/AT, the `splbio()` function is equivalent to `spl4()`.

Sleep Priorities

The second parameter to the `sleep()` function, a scheduling parameter used when the process awakes from its sleep, must be a constant and not a variable. The parameter, called the sleep priority, has critical effects on the reaction of the sleeping process to signals: if it is less than the manifest constant `PZERO` (25 on most systems); that is, if it is greater than `PZERO` (lower value priorities mean greater priority in the UNIX System), then the system does not awake sleeping processes on receipt of a signal. However, if it is less than `PZERO`, then the system awakes sleeping processes "prematurely." If the `PCATCH` bit (discussed later) is not set, the process immediately finishes the system call; that is, it executes a `longjmp()`* out of the driver.

* `sleep` calls the `longjmp()` function. When the system executes `longjmp()`, it does not follow the conventional C function call/return sequence, but instead resets the program counter, stack pointer, and data registers to the values they had when the most recent `setjmp()` function call was done.

For instance, if a signal is sent to a process sleeping in the following sleep call, the system call will end immediately without returning to the code that called sleep.

```
sleep((caddr_t)&tp->t_rawq, PZERO + 5);
```

When a driver must call sleep, how should the driver programmer determine the sleep priority? The first decision is whether the process should ignore the receipt of signals or not: if the driver puts the process to sleep for an event that is "sure" to happen, than it should ignore receipt of signals and sleep at priority greater than PZERO (numerically, less than PZERO).

An example of an event that is "sure" to happen is waiting for a locked data structure to be unlocked.

```
if (tp->t_state & T_LOCKED)
    sleep(&tp->t_state, PZERO - 5);
```

In that case, another process locked the data structure and went to sleep, but it left the data structure locked so that no other process could change it before it awoke. Since that process will eventually awake and unlock the data structure and then awake all other processes waiting for the lock to clear, the event (the wakeup call announcing the unlock) is sure to happen. Otherwise, the driver has a bug.

If the driver puts a process to sleep while it awaits an event that may not happen, the process must sleep at a priority less than PZERO (numerically greater than PZERO). An example of an event that may not happen is waiting for data to arrive from a remote device. For example, when the system reads data from a terminal, the read system call sleeps in the terminal driver waiting for data to arrive from the terminal. If data never arrives, the read will sleep indefinitely.

When a user at the terminal hits the break key or even hangs up, the terminal driver interrupt handler sends a signal to the reading process, still asleep, and the signal causes the reading process to finish the system call without having read any data. If the driver has slept at a priority value that ignores signals, the process could have been awakened only by a specific wakeup call. If that wakeup call could never happen (the user hung up the terminal), then the process would sleep forever, clearly an undesirable characteristic.

Priority values range between 0 (highest priority) and the constant PUSER (lowest system priority, usually around 60). When the driver programmer decides whether the process should ignore signals or not, he or she must choose the priority values so as not to affect process scheduling adversely. The system should be benchmarked using several sleep priority values to tune system performance with the new driver.

ADDING DEVICE DRIVERS

Drivers must occasionally "clean up" before doing the `longjmp()` on receipt of a signal while sleeping. Since the `longjmp()`, as discussed so far, takes place directly from the sleep function call, the priority parameter to the sleep function call has additional meaning: if the priority parameter is or'ed with the manifest constant `PCATCH`, the sleep call returns the value 1, if awoken on receipt of a signal. But if the sleeping process is awakened by an explicit wakeup call rather than by a signal, then the sleep call returns 0. The code sequence:

```
if (sleep(sleep_address, condition | PCATCH))
{
    /* driver code cleanup */
    .
    .
    .
    longjmp(u.u_qsav, 1);
}
```

allows the driver to clean up before doing the `longjmp()`.

The kernel saves the field `u.u_qsav` for use in a `longjmp()` function call. No other parameter should be used, nor should the driver contain a `setjmp()` function call. The second parameter of the `longjmp()` function call should always be 1.

Timeout

Sometimes, a driver arrives at a state where it wishes to reenter itself after a specified time. The driver uses the `timeout()` function for this purpose. `Timeout` takes three parameters; first, the function to be invoked when the time increment expires, second, the value of a parameter with which the function should be called, and third, the number of clock cycles to wait before the function is called. The following display is a sample timeout call.

```
timeout(repeat, n, hz)
```

where `n` is the parameter of `repeat()`, to be called after `hz` clock cycles.

If the clock interrupts the processor 60 times a second, the value of `hz` will be 60, and the system will execute the function `repeat()` in 1 second real time, as a result of the above `timeout()` call. The exact time until the `timeout` takes effect may not be precise because of the interaction of other parts of the system. The compiler requires prior declaration of the function name parameter to `timeout`, as in depending where the function `repeat()` is defined.

```
extern char *repeat();
.
.
.
timeout(repeat, n, hz);
```

In some cases, it is necessary for the driver to wait until some device operation completes. This should not be done by timing loops, especially if the driver has disabled any interrupt levels, as this would prevent other processes from running while the driver is waiting. Instead, SYSTEM V/AT provides a delay function. *Delay(n)* is called, where *n* is the number of *hz* units to delay. After $n * hz$ clock ticks the function returns. This function should only be called from the system call level, that is, not from an interrupt handler.

Allocating Buffer Space

As mentioned in the discussion on the driver *read()* and *write()* routines, drivers may require buffers for passing data around. The following utility routines in the operating system provide buffer space.

Buffer Pool

The system provides a set of buffers that are normally used for file system I/O, but they can be "borrowed" by drivers if they follow the following rules. The driver must include the header file *sys/buf.h*. The size of the buffers in the SYSTEM V/AT UNIX system is 1024 bytes. The functions that drivers may use to manipulate the buffers are listed below:

- `struct buf *getebk();`
 Allocates a buffer, and return a pointer to a buffer header that in turn, points to the data buffer.
- `brelse(bp) struct buf *bp;`
 Releases a previously allocated buffer.
- `iowait(bp) struct buf *bp;`
 Sleeps on the buffer awaiting an event, such as completion of I/O.
- `iodone(bp) struct buf *bp;`
 Awakes a process sleeping via `iowait()`.
- `clrbuf(bp) struct buf *bp;`
 Clears the contents of the buffer (set every byte in the buffer to 0) whose header is the pointer-`bp`.

The driver may access the buffer header field `b_flags` to access buffer state flags and the field `b_un.b_addr` to get the address where the data buffer resides. The following are acceptable flags to use in the `b_flags` field:

ADDING DEVICE DRIVERS

- `B_WRITE` when writing data from the buffer to the device.
- `B_READ` when reading data from the device.
- `B_DONE` set by the function `iodone()`, used to indicate that the I/O operation has completed.
- `B_ERROR` to indicate an error in use of the buffer.
- `B_BUSY` to lock the buffer and prevent other processes from accessing the buffer. Use of the `B_BUSY` flag prevents other processes from accessing the buffer, if they first check the flag to see if it is busy.

```
while (bp->b_flags & B_BUSY)
    sleep(bp, DRIPRI);
bp->b_flags |= B_BUSY;
```

- `B_WANTED` indicates that a process is sleeping, awaiting the buffer. The function `brelse()` clears the flags, `B_WANTED` and `B_BUSY`, and the function `getblk()` sets the `B_BUSY` flag. It is best to "or" and "and" the flags in, rather than just setting them, to avoid changing flag bits used by other routines.

Below is an example of the use of buffers in a tape driver.

```
tapecntl(dev, flag, opcode, arg1, arg2)
{
    register struct buf *bp;
    register int rcode;

    bp = (struct buf *) getblk();
    /* CNTL flag is used to indicate this is a control buffer */
    bp->b_flags |= B_CNTL;
    /* set async flag so buffer will be released */
    if (flag == FNDELAY)
        bp->b_flags |= B_ASYNC;
    bp->b_dev = (MTO<<8)dev;
    tapestrategy(bp);
    rcode = 0;
    if (flag != FNDELAY) {
        iowait(bp);
        if (bp->b_flags & B_ERROR)
            rcode = -1;
        bp->b_flags &= ~B_CNTL;
        brelse(bp);
    }
    return(rcode);
}
```

Clists

By including the header file *sys/tty.h*, drivers can use *clists* and *cblocks* (generic names for character lists and character blocks) to buffer small bursts of data from slow speed devices. Drivers should use clists if they are interested in character-by-character processing of data, as in terminal drivers. The only field in the cblock that a driver may access directly is shown below:

```
char c_delim
```

and the driver can use it to record status of the cblock. The size of the cblock data buffer is CLSIZE bytes, usually set between 64 and 256 bytes, compared to the buffer sizes of 1024 bytes.

The driver should not access fields in the clist or cblock data structure (except for *c_delim*) unless it uses the following routines:

- `getc(p) struct clist *p;`
Returns a character (really an int) from the clist pointed to by *p*, but it returns -1 if the clist is empty.
- `putc(c, p) char c;`
Struct *clist *p*; places the character at the end of the clist point to be *p*. If system resources are exhausted, `putc` returns -1 (error); otherwise, it returns 0.
- `struct cblock *getc()`
Returns a new cblock to the called, returning NULL if no cblocks are available in the system.
- `putc(cp) struct cblock *cp;`
Returns the cblock pointed to by *cp* to the system.
- `struct cblock *getc(p) struct clist *p;`
Returns a pointer to the first cblock on the clist *p*, but it returns NULL if the clist is empty.
- `putc(cp, p) struct cblock *cp; struct clist *p;`
Places the cblock pointed to by *cp* on the end of the clist *p*.

Here is an example of the use of clists that have been taken from a routine to read the "canonical" input from a terminal (note, the routine is not given here in its entirety). The *tty* structure contains the clists *t_canq* and *t_rawq*.

```

canon(tp)
register struct tty *tp;
{
    register struct cblock *cp;
    spl5();
    if (tp->t_rawq.c_cf == NULL)
        tp->t_delct = 0;
    while (tp->t_delct == 0) {
        if (!(tp->t_state & CARR_ON) || (u.u_fmode & FNDELAY)) {
            spl0();
            return;
        }
        tp->t_state |= IASLP;
        sleep((caddr_t)&tp->t_rawq, TTIPRI);
    }
    if (!(tp->t_flag & ICANON)) {
        yp->t_canq = tp->t_rawq;
        tp->t_rawq = ttuulq;
        tp->t_delct = 0;
        spl0();
        return;
    }
    spl0();
    while((cp = getcb(&tp->t_rawq)) != NULL) {
        putcb(cp, &tp->t_canq);
        if(cp->c_delim)
            break;
    }
    tp->t_delct--;
}

```

Dynamic Memory Allocation

In the SYSTEM V/AT system, it is possible for driver routines to allocate data space by using the kernel malloc() function. This is the preferred way to get a large buffer, as static memory allocation increases the size of the kernel image.

The kernel malloc() function takes two parameters, a memory map structure and the amount of memory to allocate from the map. Driver routines should only allocate memory from the core map. This is the same memory pool from which user processes allocate memory, so driver allocations directly reduce available memory to use processes. The driver must include the header file *sys/map.h* which defines the map structure.

The memory is allocated in map units or clicks. In the SYSTEM V/AT system a click is 512 bytes, although map unit conversion should always be done using the following macros defined in the header file *sys/sysmacros.h*:

- `ctos (c) int c;`
Converts core clicks to segments (64KB).
- `stoc (s) int s;`
Converts segments to core clicks.
- `ctod (c) int c;`
Converts core clicks to disk blocks.
- `long ctob (c) int c;`
Converts core clicks to bytes;
- `btoc (b) long b;`
Converts bytes (b) to core clicks.

The `mmap()` function cannot be called at system initialization time as the core map is undefined until after system initialization. The `mmap()` returns the click address at which the requested memory was allocated. This is a physical address and a kernel data selector must be mapped to it before it can be accessed by the driver. In the SYSTEM V/AT system the `BUFSEL` selector has been defined in the header file *sys/mmu.h* for use by drivers for dynamic memory allocation. The `mapin()` function maps the physical address returned by `mmap()` into this selector. The `mfree()` function is used to free the memory.

A trace driver might allocate a buffer for storing trace information in its `tropen()` routine. The following code segment allocates a buffer of 256 clicks (128KB), if it is available in the core map. The memory is guaranteed to be contiguous.

```
int cad;

cad = mmap (coremap, 256);
if (cad = NULL)
    u.u_error = ENOMEM;
```

In its `trwrite()` function, the trace driver can copy data from the user program into the buffer as shown in the code segment below:

```
int x;
paddr_t trp;

x = splbio();
trp = ctob ( (long) cad); /* convert to a physical address */
if (copyin (u.u_base, mapin (trp, BUFSEL), u.u_count) !=0)
    u.u_error = EFAULT;
splx(x);
```

The processor interrupt priority level should be set to mask block I/O device interrupts when BUFSEL selector is used.

In the trace driver `tread()` function, the following code segment reads the data back from the buffer:

```
x = splbio();
trp = ctob ( (long) cad);
if (copyout (mapin (trp, BUFSEL), u.u_base, u.u_count) !=0)
    u.u_error = EFAULT;
splx(x);
```

The trace driver close routine, `trclose()`, might free the trace buffer with the following code segment:

```
mfree (coremap, 256, cad);
```

The amount of memory freed, in this example, 256 clicks, must be the same amount that was allocated with the `malloc()` function.

DMA Controller Operations

The SYSTEM V/AT UNIX system implementation provides functions allowing device drives to use available DMA channels. Eight DMA channels are controlled by two INTEL™ 8237A DMA Controllers. The following table summarizes the usage of these channels in the base SYSTEM V/AT implementation:

CHANNEL	FUNCTION	Default Mode
0	Spare; 8-bit transfer	SINGLEMODE
1	Spare; 8-bit transfer	SINGLEMODE
2	Floppy controller	SINGLEMODE
3	Spare; 8-bit transfer	SINGLEMODE
4	Cascade for DMA Controller 1	CASCADEMODE
5	Spare; 16-bit transfer	SINGLEMODE
6	Spare; 16-bit transfer	SINGLEMODE
7	Spare; 16-bit transfer	SINGLEMODE

DMA channels 0, 1 and 3 are available for byte DMA operations, while channels 5, 6 and 7 are available for word transfers. To use a DMA channel the device driver must use the functions listed below:

- `opendma (channel) int channel;`

Initialize DMA channel, returns a DMA descriptor (`dmad`) if successful, or -1, if the channel is not available. This should be done when the device is opened.

- `setdma (dmd, mode, address, count)`

`int dmd;` - returned by `opendma()`.

`char mode;` - DMA controller modes are defined in `sys/8237.h`.

`paddr_t address;` - physical address to DMA to/from.

`int count;` - count of bytes/words

This function is called to set up for a DMA transfer just before initiating the device I/O. Returns 0 if successful -1 otherwise.

- `resdma (dmd) int dmd;`

Returns the residual DMA count after a transfer. This is the number of bytes/words not transferred.

- `closedma (dmd)`

This is called if no further DMA transfers will be performed, typically in the device close function.

The DMA count passed to `setdma()` and returned by `resdma()`, is an integer count of bytes for DMA channels 0-3 and of words for DMA channels 5-7. The address passed to `setdma()` must be a physical address such as that returned by

```
phys_address = physaddr (u.u_base);
```

Kernel Configuration

The SYSTEM V/AT system is configured by changing parameters in the master and *dfile* files in the *cf* directory of the Link Kit. Device driver sources are kept in the *io* directory, and the device driver object files are placed in the *lib2* archive (see AR(1)). The procedure for this is listed below:

1. Install the SYSTEM V/AT "Link Kit" and the Software Development system.
2. The Link Kit is located in the directory `usr/src/linkkit`. In the directory `usr/src/linkkit/cf` you can add a new device driver by creating new entries in the *master* and *dfile.wini* files. The *dfile.flop* file is only used for creating a floppy rooted kernel and should not contain entries for drivers not required for system installation. The *dfile* is described in CONFIG(1M) and the Master file is described in MASTER(4). Each driver in the system requires a 1 line entry in each of these files. There are also a number of turnable constants in these files.

If the driver includes an interrupt routine, the Interrupt Vector Number (IVN) used must be specified. The relationship of the IVN to the PC/AT interrupt lines is shown in the table in the section "Setting Processor Priority Levels" in this chapter.

ADDING DEVICE DRIVERS

3. Compile the new driver to be tested, and archive the objects in the lib2 archive in the *linkkit* directory with the *ar* command.
4. In the *cf* directory type the command "make" to build a winchester rooted kernel in the *linkkit* directory. This new kernel will have the filename */usr/src/linkkit/system5*. This file can be tested on the hard disk, provided the following precautions are observed:

1. Be sure you have a working boot floppy to use to restore a working kernel should the new kernel fail to boot.
2. Copy the current working kernel to a another file (*system5.last*) with the command:

```
cp /system5/ system5.last
```

5. You can test the new kernel by copying it to the root directory and rebooting (assume you are in the *cf* directory).

```
cp ../system5 /
```

```
cd /
```

press Ctl-Alt-Del to reboot

If the new kernel fails, the previous working kernel can be restored by booting the boot floppy and typing the following commands:

```
mount /dev/dsk/0s0 /mnt
```

```
cd /mnt
```

```
cp system5.last system5
```

```
cd /
```

```
umount /dev/dsk/0s0
```

```
sync
```

press Ctl-Alt-Del to reboot

TABLE OF CONTENTS OF COMMANDS

1. Commands and Application Programs, Including (1C) and (1M)

intro.....	introduction to commands and application programs
300.....	handle special functions of DASI 300 and 300s terminals
4014.....	paginator for the TEKTRONIX 4014 terminal
450.....	handles special functions of the DASI 450 terminal
accept.....	allow/prevent LP requests
admin.....	create and administer SCCS files
ar.....	archive and library maintainer for portable archives
as.....	common assembler
asa.....	interpret ASA carriage control characters
at.....	execute commands at a later time
awk.....	pattern scanning and processing language
banner.....	make posters
basename.....	deliver portions of path names
bc.....	arbitrary-precision arithmetic language
bdblk.....	print, initialize, update or recover bad sector information on disk packs
bdiff.....	big diff
bfs.....	big file scanner
brc.....	system initialization shell scripts
bs.....	a compiler/interpreter for modest-sized programs
cal.....	print calendar
calendar.....	reminder service
cat.....	concatenate and print files
cb.....	C program beautifier
cc.....	C compiler
cd.....	change working directory
cdc.....	change the delta commentary of an SCCS delta
cflow.....	generate C flow graph
checkall.....	faster file system checking procedure
chmod.....	change mode
chown.....	change owner or group
chroot.....	change root directory for a command
cli.....	clear i-node
cmp.....	compare two files
col.....	filter reverse line-feeds
comb.....	combine SCCS deltas
comm.....	select or reject lines common to two sorted files
config.....	configure a UNIX system
cp.....	copy, link or move files
cpio.....	copy file archives in and out
cpp.....	the C language preprocessor
cpset.....	install object files in binary directories
crash.....	examine system images
cron.....	clock daemon
crontab.....	user crontab file
csh.....	a shell (command interpreter) with C-like syntax
csplit.....	context split
ct.....	spawn getty to a remote terminal
chace.....	C program debugger
cu.....	call another UNIX system
cut.....	cut out selected fields of each line of a file
cxref.....	generate C program cross-reference
date.....	print and set the date
dc.....	desk calculator
dcopy.....	copy file systems for optimal access time
dd.....	convert and copy a file
delta.....	make a delta (change) to an SCCS file
devnum.....	device name
df.....	report number of free disk blocks
diff.....	differential file comparator

diff3.....3-way differential file comparison
diffmk.....mark differences between files
dircomp.....directory comparison
dis.....80286 disassembler
diskusg.....generate disk accounting data by user ID
divvy.....divide disk allocation between file systems
doscat.....concatenate and print DOS files
doscp.....copy files to and from DOS file systems
dosdir.....emulate "dir" conunand for DOS file systems
du.....summarize disk usage
dump.....dump selected parts of an object file
echo.....echo arguments
ed.....text editor
edit.....text editor (variant of ex for casual users)
efl.....Extended Fortran Language
enable.....enable/disable LP printers
env.....set environment for conunand execution
errdead.....extract error records from dump
errdemon.....error-logging daemon
erprt.....process a report of logged errors
errstop.....terminate the error-logging daemon
ex.....text editor
expr.....evaluate arguments as an expression
f77.....Fortran 77 compiler
factor.....factor a number
fdisk.....fixed disk utility
ff.....list file names and statistics for a file system
file.....determine a file type
filesave.....daily/weekly UNIX system file system backup
finc.....fast incremental backup
find.....find files
format.....format floppy and hard disk tracks
frec.....recover files from a backup device
fsck.....file system consistency check and interactive repair
fsdb.....file system debugger
fsplit.....split f77, ratfor, or efl files
fsstat.....file system status
fuser.....identify processes using a file or file structure
fwtinp.....manipulate connect accounting records
get.....get a version of an SCCS file
getopt.....parse conunand options
getty.....set terminal type, modes, speed and line discipline
greek.....select terminal filter
grep.....search a file for a pattern
help.....ask for help
hp.....handle special functions of Hewlett-Packard 2640 and 2621-series terminals
hpio.....Hewlett-Packard 2645A terminal tape file archiver
hyphen.....find hyphenated words
ib.....install boot image
id.....print user and group IDs and names
init.....process control initialization
init-inittab.....script for the init process
install.....install commands
installit.....package installation
ipcrm.....remove a message queue, semaphore set or shared memory id
ipcs.....report inter-process conununication facilities status
join.....relational database operator
keyset.....programunable function keys
kill.....terminate a process
killall.....kill all active processes
ld.....link editor for conunon object files
lex.....generate programs for simple lexical tasks

line.....read one line
link.....exercise link and unlink system calls
lint.....a C program checker
login.....sign on
logname.....get login name
lorder.....find ordering relation for an object library
lp.....send/cancel requests to an LP line printer
lpadmin.....configure the LP spooling system
lpget,lpset.....initialize the parallel printer driver
lpsched.....start/stop the LP request scheduler and move requests
lpstat.....print LP status information
ls.....list contents of directory
m4.....macro processor
machid.....provide truth value about your processor type
mail.....send mail to users or read mail
mailx.....interactive message processing system
make.....maintain, update, and regenerate groups of programs
man.....print entries in this manual
mesg.....permit or deny messages
mkdir.....make a directory
mkfs.....construct a file system
mknod.....build a special file
mount.....mount and dismount file system
mvdir.....move a directory
ncheck.....generate names from i-numbers
newform.....change the format of a text file
newgrp.....log in to a new group
news.....print news items
nice.....run a command at low priority
nl.....line numbering filter
nm.....print name list of conunon object file
nodename.....change or display system node name
nohup.....run a command inunune to hangups and quits
od.....octal dump
pack.....compress and expand files
passwd.....change login password
passwd-passwd file format.....password file
paste.....merge same lines of several files or subsequent lines of one file
patch.....inspect or modify an STL- or COFF-format binary file
pg.....file perusal filter for soft-copy terminals
pr.....print files
prof.....display profile data
profiler.....operating system profiler
prs.....print an SCCS file
ps.....report process status
ptx.....permuted index
pwck.....password/group file checkers
pwd.....working directory name
raffor.....rational Fortran dialect
regcmp.....regular expression compile
rjstat.....RJE status report and interactive status console
rm.....remove files or directories
rmdel.....remove a delta from an SCCS file
sact.....print current SCCS file editing activity
sadb.....disk access profiler
sar.....system activity reporter
sccsdiff.....compare two versions of an SCCS file
sdb.....symbolic debugger
sdiff.....side-by-side difference program
sed.....stream1 editor
send.....gather files and/or submit RJE jobs
setcolor.....set foreground and background colors

setmnt.....establish mount table
 setup.....define device characteristics
 sh.....shell, the standard/restricted command programming language
 shl.....shell layer manager
 shmcreate.....user-mode access to console graphics
 showbad.....display bad track table for hard disk partition
 shutdown.....terminate all processing
 size.....print section sizes of common object files
 sleep.....suspend execution for an interval
 sno.....SNOBOL interpreter
 sort.....sort and/or merge files
 spell.....find spelling errors
 split.....split a file into pieces
 silboot.....system load bootstrap program
 strip.....strip symbol and line number information from a common object file
 stty.....set the options for a terminal
 su.....become super-user or another user
 sum.....print checksum and block count of a file
 sync.....update the super block
 sysdef.....system definition
 tabs.....set tabs on a terminal
 tail.....deliver the last part of a file
 tar.....tape file archiver
 tee.....pipe fitting
 test.....condition evaluation command
 tic.....terminfo compiler
 time.....time a command
 timex.....time a command; report process data and system activity
 touch.....update access and modification times of a file
 tput.....query terminfo for database
 tr.....translate characters
 true.....provide true values
 tsort.....topological sort
 tty.....get the name of the terminal
 ttypatch.....patch a kernel for tty parameters
 uadmin.....administrative control
 umask.....set file-creation mode mask
 uname.....print name of current UNIX system
 unget.....undo a previous get of an SCCS file
 uniq.....report repeated lines in a file
 units.....conversion program
 untc.....uncompile terminfo terminal description files
 uuclean.....uucp spool directory clean-up
 uucp.....UNIX system to UNIX system copy
 uustat.....uucp status inquiry and job control
 uusuh.....monitor uucp network
 uuto.....public UNIX-to-UNIX system file copy
 uux.....UNIX-to-UNIX system command execution
 val.....validate SCCS file
 vc.....version control
 vi.....screen-oriented (visual) display editor based on ex
 volcopy.....copy file systems with label checking
 vprmsave.....save and print VPM event traces
 wait.....await completion of process
 wall.....write to all users
 wc.....word count
 what.....identify SCCS files
 who.....who is on the system
 whodo.....who is doing what
 write.....write to another user
 xargs.....construct argument list(s) and execute command
 yacc.....yet another compiler-compiler

The *300* command can be necessary to insert page number. Instead of hitting the feed key to get any response.

DES

In many (but not all) cases

nroff -T300 files
nroff -T300-12

The use of *300* can thus be required; in a few cases, it may produce better alignment.

The *neqn* names of, and supported by *300* are shown

SEE ALSO

450(1), *graph*(1G), *mesg*(1),
"Nroff and Troff User Manual",
"Formatting Program" (tbl)

BUGS

Some special characters on the print head cannot be moved. If your output contains a carriage return, it has a tendency to slip when misaligning the first line of

NAME

intro - introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, publicly-accessible commands. Certain distinctions of purpose are made in the headings:

- (1) Commands of general utility.
- (1C) Commands for communication with other systems.
- (1M) System maintenance commands (may be restricted to superuser)

COMMAND SYNTAX

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option*(*s*)] [*cmdarg*(*s*)]

where:

name The name of an executable file.

option - *noargletter*(*s*) or,
- *argletter*<>*optarg*
where <> is optional white space.

noargletter A single letter representing an option without an argument.

argletter A single letter representing an option requiring an argument.

optarg Argument (character string) satisfying preceding *argletter*.

cmdarg Path name (or other command argument) *not* beginning with - or, - by itself indicating the standard input.

SEE ALSO

getopt(1),
exit(2), *wait*(2), *getopt*(3C) in the Software Development System manual.
"Introduction" and "Unix System Capabilities" at the front of this volume.

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program [see *wait*(2) and *exit*(2)]. The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

BUGS

Regretfully, many commands do not adhere to the aforementioned syntax.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.

300(1)

NAME

300, 300s - handle special

SYNOPSIS

300 [+12] [-n] [-
300s [+12] [-n] [

DESCRIPTION

The 300 command supports
300 (GSI 300 or DTC 300),
DASI 300s (GSI 300s or
half-line reverse, and full-
It also attempts to draw a
convenient use of 12-pitch
300 command can be used

neqn file ... | nroff

WARNING: if your terminal
before 300 is used.

The behavior of 300 can be
12-pitch text, fractional line

+12 permits use of 12
mally allow only
pitch, 8 lines/inch
tion, the user should
option.

-n controls the size of
to 4 vertical plot
an inch, a 10-pitch
line-feed needs of
value, thus allowing
scripts and superscripts
to act as quarter
appropriate half-line
option -3 alone,)

-dt,l,c controls delay factor
terminals sometimes
long lines, too many
identical character
for every set of t
blank, non-tab character
length)/20 nulls are
omitted from the e
Also, a value of zero
acter). The former
files like /etc/passwd
the specific character
may have to experiment
-d option exists on
otherwise print program
printed using **-d3**,
programs that have

Note that the delay
riage return and line
cr3 are recommended

ACCEPT(1M)

NAME

accept, reject - allow/prevent LP requests

SYNOPSIS

/usr/lib/accept destinations
/usr/lib/reject [-r[reason]] destinations

DESCRIPTION

Accept allows *lp(1)* to accept requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat(1)* to find the status of *destinations*.

Reject prevents *lp(1)* from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat(1)* to find the status of *destinations*. The following option is useful with *reject*.

-r[reason] Associates a *reason* with preventing *lp* from accepting requests. This *reason* applies to all printers mentioned up to the next **-r** option. *Reason* is reported by *lp* when users direct requests to the named *destinations* and by *lpstat(1)*. If the **-r** option is not present or the **-r** option is given without a *reason*, then a default *reason* will be used.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1).

NAME

`admin` - create and administer SCCS files

SYNOPSIS

```
admin [-n] [-i{name}] [-rrel] [-t{name}] [-fflag{flag-val}]
[-dflag{flag-val}] [-alogin] [-elogin] [-m{mrlist}] [-y{comment}] [-h]
[-z] files
```

DESCRIPTION

`Admin` is used to create new SCCS files and change parameters of existing ones. Arguments to `admin`, which may appear in any order, consist of keyletter arguments, which begin with -, and named files (note that SCCS file names must begin with the characters `s.`). If a named file does not exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, `admin` behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with `s.`) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

- n This keyletter indicates that a new SCCS file is to be created.
- i{name} The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see -r keyletter for delta numbering scheme). If the `i` keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an `admin` command on which the `i` keyletter is supplied. Using a single `admin` to create two or more SCCS files requires that they be created empty (no -i keyletter). Note that the -i keyletter implies the -n keyletter.
- rrel The *release* into which the initial delta is inserted. This keyletter may be used only if the -i keyletter is also used. If the -r keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).
- t{name} The *name* of a file from which descriptive text for the SCCS file is to be taken. If the -t keyletter is used and `admin` is creating a new SCCS file (the -n and/or -i keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a -t keyletter without a file name causes removal of descriptive text (if any)

currently in the SCCS file, and (2) a `-t` keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

- `-f flag` This keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several *f* keyletters may be supplied on a single *admin* command line. The allowable *flags* and their values are:
- b** Allows use of the `-b` keyletter on a *get(1)* command to create branch deltas.
 - cceil** The highest release (i.e., "ceiling"), a number less than or equal to 9999, which may be retrieved by a *get(1)* command for editing. The default value for an unspecified *c* flag is 9999.
 - ffloor** The lowest release (i.e., "floor"), a number greater than 0 but less than 9999, which may be retrieved by a *get(1)* command for editing. The default value for an unspecified *f* flag is 1.
 - dsid** The default delta number (SID) to be used by a *get(1)* command.
 - i[str]** Causes the "No id keywords (ge6)" message issued by *get(1)* or *delta(1)* to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords [see *get(1)*] are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must exactly match the given string, however the string must contain a keyword, and no embedded newlines.
 - j** Allows concurrent *get(1)* commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.
 - l*list*** A *list* of releases to which deltas can no longer be made (*get -e* against one of these "locked" releases fails). The *list* has the following syntax:

```
<list> ::= <range> | <list> , <range>
<range> ::= RELEASE NUMBER | a
```

 The character *a* in the *list* is equivalent to specifying *all releases* for the named SCCS file.
 - n** Causes *delta(1)* to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.
 - qt*ext*** User definable text substituted for all occurrences of the `%Q%` keyword in SCCS file text retrieved by *get(1)*.

- mmod** *Module name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by get(1). If the m flag is not specified, the value assigned is the name of the SCCS file with the leading s. removed.*
- tttype** *Type of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by get(1).*
- v[pgm]** *Causes delta(1) to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validity checking program [see delta(1)]. (If this flag is set when creating an SCCS file, the m keyletter must also be used even if its value is null).*
- dflag** *Causes removal (deletion) of the specified flag from an SCCS file. The -d keyletter may be specified only when processing existing SCCS files. Several -d keyletters may be supplied on a single admin command. See the -f keyletter for allowable flag names.*
- list** *A list of releases to be "unlocked". See the -f keyletter for a description of the l flag and the syntax of a list.*
- alogin** *A login name, or numerical UNIX system group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all login names common to that group ID. Several a keyletters may be used on a single admin command line. As many logins, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If login or group ID is preceded by a ! they are to be denied permission to make deltas.*
- elogin** *A login name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all login names common to that group ID. Several e keyletters may be used on a single admin command line.*
- y[comment]** *The comment text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of delta(1). Omission of the -y keyletter results in a default comment line being inserted in the form:*
- date and time created YY/MM/DD HH:MM:SS by login*
- The -y keyletter is valid only if the -i and/or -n keyletters are specified (i.e., a new SCCS file is being created).*
- m[mrlist]** *The list of Modification Requests (MR) numbers is inserted into the SCCS file as the reason for creating*

ADMIN(1)

the initial delta in a manner identical to *delta(1)*. The **v** flag must be set and the *MR* numbers are validated if the **v** flag has a value (the name of an *MR* number validation program). Diagnostics will occur if the **v** flag is not set or *MR* validation fails.

-h Causes *admin* to check the structure of the SCCS file [see *sccsfile(5)*], and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

-z The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see **-h**, above).

Note that use of this keyletter on a truly corrupted file may prevent future detection of the corruption.

FILES

The last component of all SCCS file names must be of the form *s,file-name*. New SCCS files are given mode 444 [see *chmod(1)*]. Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file, called *x,file-name*, [see *get(1)*], created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of *ed(1)*. *Care must be taken!* The edited file should *always* be processed by an **admin -h** to check for corruption followed by an **admin -z** to generate a proper check-sum. Another **admin -h** is recommended to ensure the SCCS file is valid.

Admin also makes use of a transient lock file (called *x,file-name*), which is used to prevent simultaneous updates to the SCCS file by different users. See *get(1)* for further information.

SEE ALSO

delta(1), *ed(1)*, *get(1)*, *help(1)*, *prs(1)*, *what(1)*,
sccsfile(4) in the Software Development System manual.

Source Code Control System User Guide in the Software Development System manual.

DIAGNOSTICS

Use *help(1)* for explanations.

NAME

`ar` - archive and library maintainer for portable archives

SYNOPSIS

`ar` *key* [*posname*] *afile* name ...

DESCRIPTION

Ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose.

When *ar* creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure is described in detail in *ar*(4). The archive symbol table [described in *ar*(4)] is used by the link editor [*ld*(1)] to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by *ar* when there is at least one object file in the archive. The archive symbol table is in a specially named file which is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the *ar* command is used to create or update the contents of an archive, the symbol table is rebuilt. The symbol table can be forced to be rebuilt by the *s* option described below.

Key is an optional -, followed by one character from the set *drqtpmx*, optionally concatenated with one or more of *vuaibcls*. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

- d** Delete the named files from the archive file.
- r** Replace the named files in the archive file. If the optional character *u* is used with *r*, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set *abi* is used, then the *posname* argument must be present and specifies that new files are to be placed after (*a*) or before (*b* or *i*) *posname*. Otherwise, new files are placed at the end.
- q** Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. This command is useful only to avoid quadratic behavior when creating a large archive piece-by-piece.
- t** Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.
- p** Print the named files in the archive.
- m** Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in *r*, specifies where the files are to be moved.
- x** Extract the named files. If no names are given, all files in the archive are extracted. In neither case does *x* alter the archive file.
- v** Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with *t*, it gives a long listing of all information about the files. When used with *x*, it precedes each file with a name.
- c** Suppress the message that is produced by default when *afile* is created.

AR(1)

- l** Place temporary files in the local current working directory, rather than in the directory specified by the environment variable *TMPDIR* or in the default directory */tmp*.
- s** Force the regeneration of the archive symbol table even if *ar* is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the *strip*(1) command has been used on the archive.

FILES

*/tmp/ar** temporaries

SEE ALSO

ld(1), *lorder*(1), *strip*(1),
tmpnam(3S), *a.out*(4), *ar*(4) in the Software Development System manual.

BUGS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

NAME

as - assembler

SYNOPSIS

as [-o objfile] [-n] [-m] [-R] [-V] [-u] [-Ms] [-Ml] file-name

DESCRIPTION

The *as* command assembles the named file. The following options may be specified in any order:

- o *objfile* Put the output of the assembly in *objfile*. By default, the output file name is formed by removing the *.s* suffix, if there is one, from the input file name and appending a *.o* suffix.
- n Turn off long/short address optimization. By default, address optimization takes place.
- m Run the *m4* macro preprocessor on the input to the assembler.
- R Remove (unlink) the input file after assembly is completed.
- V Write the version number of the assembler being run on the standard error output.
- u Remove unreferenced debugging symbols from the output symbol table. This option may be used in conjunction with the *-g* option of *cc(1)*. This will make object files smaller and decrease the debugger startup time, but will require more time to assemble the file.
- Ms, -Ml Generate small model object code (default) and generate large model object code, respectively. This option is used simply to pass the correct magic number to the loader via the UNIX system header. Actual code assembled is not affected.

FILES

/usr/tmp/as[1-6]XXXXXX temporary files

SEE ALSO

ld(1), m4(1), mn(1), strip(1).
a.out(4) in the Software Development System manual.

WARNING

If the input file does not contain a *.file* assembler directive and the *-m* flag was specified, the file name given by the assembler when an error occurs is one of the temporary files (*/usr/tmp/asXXXXXX*).

If the *m4* macro preprocessor (the *-m* option) is used, keywords for *m4* [see *m4(1)*] cannot be used as symbols (variables, functions, labels) in the input file since *m4* cannot determine which are assembler symbols and which are real *m4* macros.

BUGS

The *.even* assembler directive is not guaranteed to work in the *.text* section when optimization is performed.

Arithmetic expressions may only have one forward referenced symbol per expression.

ASA(1)

NAME

`asa` -- interpret ASA carriage control characters

SYNOPSIS

`asa` [*files*]

DESCRIPTION

Asa interprets the output of FORTRAN programs that utilize ASA carriage control characters. It processes either the *files* whose names are given as arguments or the standard input if no file names are supplied. The first character of each line is assumed to be a control character; their meanings are:

- ' ' (blank) single new line before printing
- 0 double new line before printing
- 1 new page before printing
- + overprint previous line.

Lines beginning with other than the above characters are treated as if they began with ' '. The first character of a line is *not* printed. If any such lines appear, an appropriate diagnostic will appear on standard error. This program forces the first line of each input file to start on a new page.

To view correctly the output of FORTRAN programs which use ASA carriage control characters, *asa* could be used as a filter thus:

```
a.out | asa | lp
```

and the output, properly formatted and paginated, would be directed to the line printer. FORTRAN output sent to a file could be viewed by:

```
asa file
```

SEE ALSO

`efl(1)`, `f77(1)`, `fsplit(1)`, `ratfor(1)`.

NAME

at, *batch* — execute commands at a later time

SYNOPSIS

```
at time [ date ] [ + increment ]
at -r job ...
at -l [ job ... ]

batch
```

DESCRIPTION

At and *batch* read commands from standard input to be executed at a later time. *At* allows you to specify when the commands should be executed, while jobs queued with *batch* will execute when system load level permits. The *-r* option removes jobs previously scheduled with *at*. The *-l* option reports all jobs scheduled for the invoking user.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, *umask*, and *ulimit* are retained when the commands are executed. Open file descriptors, traps, and priority are lost.

Users are permitted to use *at* if their names appear in the file */usr/lib/cron/at.allow*. If that file does not exist, the file */usr/lib/cron/at.deny* is checked to determine if the user should be denied access to *at*. If neither file exists, only root is allowed to submit a job. If either file is *at.deny*, global usage is permitted. The allow/deny files consist of one user name per line.

The *time* may be specified as 1, 2, or 4 digits. One and two-digit numbers are taken to be hours, four digits to be hours and minutes. The time may alternately be specified as two numbers separated by a colon, meaning *hour:minute*. A suffix *am* or *pm* may be appended; otherwise a 24-hour clock time is understood. The suffix *zulu* may be used to indicate GMT. The special names *noon*, *midnight*, *now*, and *next* are also recognized.

An optional *date* may be specified as either a month name followed by a day number (and possibly year number preceded by an optional comma) or a day of the week (fully spelled or abbreviated to three characters). Two special "days", *today* and *tomorrow* are recognized. If no *date* is given, *today* is assumed if the given hour is greater than the current hour and *tomorrow* is assumed if it is less. If the given month is less than the current month (and no year is given), *next* year is assumed.

The optional *increment* is simply a number suffixed by one of the following: **minutes**, **hours**, **days**, **weeks**, **months**, or **years**. (The singular form is also accepted.)

Thus legitimate commands include:

```
at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at 5 pm Friday
```

At and *batch* write the job number and schedule time to standard error.

Batch submits a batch job. It is almost equivalent to "at now", but not quite. For one, it goes into a different queue. For another, "at now" will respond with the error message too late.

The *-r* option removes jobs previously scheduled by *at* or *batch*. The job number is the number given to you previously by the *at* or *batch* command. You can also get job numbers by typing *at -l*. You can only remove your own jobs unless you are the super-user.

AT(1)

EXAMPLES

The *at* and *batch* commands read from standard input the commands to be executed at a later time. *Sh(1)* provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
batch
nroff filename >outfile
<control-D> (hold down 'control' and depress 'D')
```

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

```
batch <<!
nroff filename 2>&1 >outfile | mail loginid
!
```

To have a job reschedule itself, invoke *at* from within the shell procedure, by including code similar to the following within the shell file:

```
echo "sh shellfile" | at 1900 thursday next week
```

FILES

<i>/usr/lib/cron</i>	main cron directory
<i>/usr/lib/cron/at.allow</i>	list of allowed users
<i>/usr/lib/cron/at.deny</i>	list of denied users
<i>/usr/lib/cron/queue</i>	scheduling information
<i>/usr/spool/cron/atjobs</i>	spool area

SEE ALSO

kill(1), *mail(1)*, *nice(1)*, *ps(1)*, *sh(1)*.
cron(1M).

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

awk — pattern scanning and processing language

SYNOPSIS

awk [*-F* *c*] [*prog*] [*parameters*] [*files*]

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as *-f file*. The *prog* string should be enclosed in single quotes (') to protect it from the shell.

Parameters, in the form *x=...* *y=...* etc., may be passed to *awk*.

Files are read in order; if there are no files, the standard input is read. The file name *-* means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using *FS*; see below). The fields are denoted *\$1*, *\$2*, ...; *\$0* refers to the entire line.

A pattern-action statement has the form:

```
pattern { action }
```

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next # skip remaining patterns on this input line
exit # skip the rest of the input
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the operators *+*, *-*, ***, */*, *%*, and concatenation (indicated by a blank). The C operators *++*, *--*, *+=*, *-=*, **=*, */=*, and *%=* are also available in expressions. Variables may be scalars, array elements (denoted *x[i]*) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (").

The *print* statement prints its arguments on the standard output (or on a file if *>expr* is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its expression list according to the format [see *printf*(3S)].

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp*, *log*, *sqrt*, and *int*. The last truncates its argument to an integer; *substr*(*s*, *m*, *n*) returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf*(*fmt*, *expr*, *expr*, ...) formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

AWK(1)

Patterns are arbitrary Boolean combinations (! , | , & & , and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep* [see *grep*(1)]. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

```
expression matchop regular-expression
expression relop expression
```

where a relop is any of the six relational operators in C, and a matchop is either ~ (for *contains*) or !~ (for *does not contain*). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character *c* may be used to separate the fields by starting the program with:

```
BEGIN { FS = c }
```

or by using the -F*c* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default new-line); and OFMT, the output format for numbers (default %.6g).

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

```
{ s += $1 }
END { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }
{ print }
```

command line: `awk -f program n=5 input`

SEE ALSO

grep(1), lex(1), sed(1).
malloc(3X) in the Software Development System manual.

BUGS

Input white space is not preserved on output if fields are involved. There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

BANNER(1)

NAME

banner — make posters

SYNOPSIS

banner strings

DESCRIPTION

Banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

SEE ALSO

echo(1).

NAME

basename, dirname — deliver portions of path names

SYNOPSIS

```
basename string [ suffix ]
dirname string
```

DESCRIPTION

Basename deletes any prefix ending in / and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside substitution marks () within shell procedures.

Dirname delivers all but the last level of the path name in *string*.

EXAMPLES

The following example, invoked with the argument `/usr/src/cmd/cat.c`, compiles the named file and moves the output to a file named `cat` in the current directory:

```
cc $1
mv a.out basename $1 \c
```

The following example will set the shell variable `NAME` to `/usr/src/cmd`:

```
NAME=dirname /usr/src/cmd/cat.c
```

SEE ALSO

sh(1).

BUGS

The *basename* of / is null and is considered an error.

BC(1)

NAME

bc — arbitrary-precision arithmetic language

SYNOPSIS

bc [-c] [-l] [file ...]

DESCRIPTION

Bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The `-l` argument stands for the name of an arbitrary-precision math library. The syntax for *bc* programs is as follows; L means letter a-z, E means expression, S means statement.

Comments

are enclosed in `/*` and `*/`.

Names

simple variables: L

array elements: L [E]

The words "ibase", "obase", and "scale"

Other operands

arbitrarily long numbers with optional sign and decimal point.

(E)

sqrt (E)

length (E) number of significant decimal digits

scale (E) number of digits right of decimal point

L (E , ... , E)

Operators

+ - * / % ^ (% is remainder; ^ is power)

++ -- (prefix and postfix; apply to names)

== <= >= != < >

== + == - == * == / == % == ^

Statements

E

{ S ; ... ; S }

if (E) S

while (E) S

for (E ; E ; E) S

null statement

break

quit

Function definitions

define L (L , ... , L) {

 auto L , ... , L

 S ; ... S

 return (E)

}

Functions in `-l` math library

s(x) sine

c(x) cosine

e(x) exponential

l(x) log

a(x) arctangent

j(n,x) Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or new-lines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc(1)*. Assignments to *ibase* or *obase* set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. "Auto" variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

Bc is actually a preprocessor for *dc(1)*, which it invokes automatically, unless the *-c* (compile only) option is present. In this case the *dc* input is sent to the standard output instead.

EXAMPLE

```
scale = 20
define e(x){
    auto a, b, c, i, s
    a = 1
    b = 1
    s = 1
    for(i=1; i==1; i++){
        a = a*x
        b = b*i
        c = a/b
        if(c == 0) return(s)
        s = s+c
    }
}
```

defines a function to compute an approximate value of the exponential function and

```
for(i=1; i<=10; i++) e(i)
```

prints approximate values of the exponential function of the first ten integers.

FILES

```
/usr/lib/lib.b    mathematical library
/usr/bin/dc      desk calculator proper
```

SEE ALSO

dc(1).

BUGS

No *&&*, *||* yet.

For statement must have all three E's.

Quit is interpreted when read, not when executed.

BDBLK(1)

NAME

`bdblk` - print, initialize, update or recover bad sector information on disk packs

SYNOPSIS

`/etc/bdblk` option unit [sector ...]

DESCRIPTION

`Bdblk` can be used to print, initialize, update or recover the bad block information stored on disk that is used by the disk drivers to implement bad sector replacement.

The bad sector information on 3B20 computers is located in the last sector of the first cylinder of the disk pack. The bad sector information on DEC is located in the last sector of the last cylinder of the disk pack. The bad sector information on Intel 286 systems is located in the last sector of the <<T B D>> cylinder of the disk pack.

Replacement sectors are allocated starting with the first sector before the bad sector information and working backward toward the beginning of the disk. A maximum of 126 bad sectors are supported. The position of the bad sector in the bad sector table determines which replacement sector it corresponds to.

The bad sector information structure is as follows:

```
struct badblk {
    int bb_magic;          /* bad block information magic number */
    int bb_count;         /* number of bad sectors in table */
    daddr bb_blkno[126]; /* sector number of bad sector */
};
```

`Bdblk` is invoked by giving an *option* and the *unit* number of the disk drive number. The *option* is specified by one of the following letters:

- p** It reads the bad sector information from the specified unit and prints out the bad sector information.
- i** It verifies the format of the specified unit and initializes the bad sector information on disk.
- u** It verifies the format of the specified unit and updates the bad sector information on disk.
- r** It may be invoked by giving a list of bad sectors. It will then write the supplied information onto the disk. This option should only be used to restore known bad sector information which was destroyed.

SEE ALSO

`format(1M)`, `wn(7)`.

WARNINGS

After having changed the bad sector information on disk, the disk should be put out of service to insure the system bad block information table for that unit is current.

NAME**bdiff** -- big diff**SYNOPSIS****bdiff** file1 file2 [n] [-s]**DESCRIPTION**

Bdiff is used in a manner analogous to *diff*(1) to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for *diff*. *Bdiff* ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. The value of *n* is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail. If *file1* (*file2*) is -, the standard input is read. The optional -s (silent) argument specifies that no diagnostics are to be printed by *bdiff* (note, however, that this does not suppress possible exclamations by *diff*). If both optional arguments are specified, they must appear in the order indicated above.

The output of *bdiff* is exactly that of *diff*, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

FILES

/tmp/bd????

SEE ALSO

diff(1).

DIAGNOSTICSUse *help*(1) for explanations.

BFS(1)

NAME

bfs - big file scanner

SYNOPSIS

bfs [-] name

DESCRIPTION

The *Bfs* command is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes (the maximum possible size) and 32K lines, with up to 512 characters, including new-line, per line (255 for 16-bit machines). *Bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit*(1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the *w* command. The optional *-* suppresses printing of sizes. Input is prompted with *** if *P* and a carriage return are typed as in *ed*. Prompting can be turned off again by inputting another *P* and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols besides */* and *?*: *>* indicates downward search without wrap-around, and *<* indicates upward search without wrap-around. There is a slight difference in mark names: only the letters *a* through *z* may be used, and all 26 marks are remembered.

The *e*, *g*, *v*, *k*, *p*, *q*, *w*, *=*, *!* and null commands operate as described under *ed*. Commands such as *--*, *+++*, *+++=*, *-12*, and *+4p* are accepted. Note that *1,10p* and *1,10* will both print the first ten lines. The *f* command only prints the name of the file being scanned; there is no *remembered* file name. The *w* command is independent of output diversion, truncation, or crunching (see the *xo*, *xt* and *xc* commands, below). The following additional commands are available:

xf file

Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the *xf*. The *xf* commands may be nested to a depth of 10.

xn List the marks currently in use (marks are set by the *k* command).

xo [file]

Further output from the *p* and null commands is diverted to the named *file*, which, if necessary, is created mode 666. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: label

This positions a *label* in a command file. The *label* is terminated by new-line, and blanks between the *:* and the start of the *label* are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(...)xb/regular expression/label

A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between 1 and \$.
2. The second address is less than the first.
3. The regular expression does not match at least one line in the specified range, including the first and last lines.

On success, . is set to the line matched and a jump is made to *label*. This command is the only one that does not issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

```
xb/^/ label
```

is an unconditional jump.

The **xb** command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe only a downward jump is possible.

xt *number*

Output from the **p** and null commands is truncated to at most *number* characters. The initial number is 255.

xv[*digit*][*spaces*][*value*]

The variable name is the specified *digit* following the xv. The commands **xv5100** or **xv5 100** both assign the value **100** to the variable **5**. The command **Xv61,100p** assigns the value **1,100p** to the variable **6**. To reference a variable, put a % in front of the variable name. For example, using the above assignments for variables **5** and **6**:

```
l,%5p
l,%5
%6
```

will all print the first 100 lines.

```
g/%5/p
```

would globally search for the characters **100** and print each line containing a match. To escape the special meaning of %, a \ must precede it.

```
g/".*\%[cds]/p
```

could be used to match and list lines containing *printf* of characters, decimal integers, or strings.

Another feature of the **xv** command is that the first line of output from a UNIX system command can be stored into a variable. The only requirement is that the first character of *value* be an !. For example:

```
.w junk
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

would put the current line into variable **5**, print it, and increment the variable **6** by one. To escape the special meaning of ! as the first character of *value*, precede it with a \.

BFS(1)

xv7\!date

stores the value *!date* into variable 7.

xbz label

xbn label

These two commands will test the last saved *return code* from the execution of a UNIX system command (*!command*) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string *size*.

```
xv55
:1
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn 1
xv45
:1
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz 1
```

xc [*switch*]

If *switch* is 1, output from the *p* and null commands is crunched; if *switch* is 0 it is not. Without an argument, *xc* reverses *switch*. Initially *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

SEE ALSO

csplit(1), *ed(1)*.

regcmp(3X) in the Software Development System manual.

DIAGNOSTICS

? for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.

NAME

brc, bcheckrc, rc, powerfail — system initialization shell scripts

SYNOPSIS

/etc/brc

/etc/bcheckrc

/etc/rc

/etc/powerfail

DESCRIPTION

Except for *powerfail*, these shell procedures are executed via entries in *etc/inittab* by *init(1M)* when the system is changed out of *SINGLE USER* mode. *Powerfail* is executed whenever a system power failure is detected.

The *brc* procedure clears the mounted file system table, */etc/mnttab* [see *mnttab(4)*], and loads any programmable microprocessors with their appropriate scripts.

The *bcheckrc* procedure performs all the necessary consistency checks to prepare the system to change into multiuser mode. It will prompt to set the system date and to check the file systems with *fsc(1M)*.

The *rc* procedure starts all system daemons before the terminal lines are enabled for multiuser mode. In addition, file systems are mounted and accounting, error logging, system activity logging and the Remote Job Entry (RJE) system are activated in this procedure.

The *powerfail* procedure is invoked when the system detects a power failure condition. Its chief duty is to reload any programmable microprocessors with their appropriate scripts, if suitable. It also logs the fact that a power failure occurred.

SEE ALSO

fsc(1M), *init(1M)*, *shutdown(1M)*, *inittab(4)*, *mnttab(4)*.

BS(1)

NAME

bs — a compiler/interpreter for modest-sized programs

SYNOPSIS

bs [file [args]]

DESCRIPTION

Bs is a remote descendant of Basic and Snobol4 with a little C language thrown in. *Bs* is designed for programming tasks where program development time is as important as the resulting speed of execution. Formalities of data declaration and file/process manipulation are minimized. Line-at-a-time debugging, the *trace* and *dump* statements, and useful run-time error messages all simplify program testing. Furthermore, incomplete programs can be debugged; *inner* functions can be tested before *outer* functions have been written and vice versa.

If the command line *file* argument is provided, the file is used for input before the console is read. By default, statements read from the file argument are compiled for later execution. Likewise, statements entered from the console are normally executed immediately (see *compile* and *execute* below). Unless the final operation is assignment, the result of an immediate expression statement is printed.

Bs programs are made up of input lines. If the last character on a line is a \, the line is continued. *Bs* accepts lines of the following form:

```
statement
label statement
```

A label is a *name* (see below) followed by a colon. A label and a variable can have the same name.

A *bs* statement is either an expression or a keyword followed by zero or more expressions. Some keywords (*clear*, *compile*, *!*, *execute*, *include*, *ibase*, *obase*, and *run*) are always executed as they are compiled.

Statement Syntax:

expression

The expression is executed for its side effects (value, assignment, or function call). The details of expressions follow the description of statement types below.

break

Break exits from the inner-most *forwhile* loop.

clear

Clears the symbol table and compiled statements. *Clear* is executed immediately.

compile [expression]

Succeeding statements are compiled (overrides the immediate execution default). The optional expression is evaluated and used as a file name for further input. A *clear* is associated with this latter case. *Compile* is executed immediately.

continue

Continue transfers to the loop-continuation of the current *forwhile* loop.

dump [name]

The name and current value of every non-local variable is printed. Optionally, only the named variable is reported. After an error or interrupt, the number of the last statement and (possibly) the user-function trace are displayed.

exit [expression]

Return to system level. The expression is returned as process status.

execute

Change to immediate execution mode (an interrupt has a similar effect).

This statement does not cause stored statements to execute (see *run* below).

for name = expression expression statement

for name = expression expression

...

next

for expression , expression , expression statement

for expression , expression , expression

...

next

The *for* statement repetitively executes a statement (first form) or a group of statements (second form) under control of a named variable. The variable takes on the value of the first expression, then is incremented by one on each loop, not to exceed the value of the second expression. The third and fourth forms require three expressions separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action (normally an increment).

fun f([a, ...]) [v, ...]

...

nuf

Fun defines the function name, arguments, and local variables for a user-written function. Up to ten arguments and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions may not be nested.

freturn

A way to signal the failure of a user-written function. See the interrogation operator (?) below. If interrogation is not present, *freturn* merely returns zero. When interrogation is active, *freturn* transfers to that expression (possibly by-passing intermediate function returns).

goto name

Control is passed to the internally stored statement with the matching label.

ibase *N*

Ibase sets the input base (radix) to *N*. The only supported values for *N* are 8, 10 (the default), and 16. Hexadecimal values 10-15 are entered as a-f.

A leading digit is required (i.e., f0a must be entered as 0f0a). *Ibase* (and *obase*, below) are executed immediately.

if expression statement

if expression

...

[**else**

...]

fi

The statement (first form) or group of statements (second form) is executed if the expression evaluates to non-zero. The strings 0 and "" (null) evaluate as zero. In the second form, an optional *else* allows for a group of statements to be executed when the first group is not. The only statement permitted on the same line with an *else* is an *if*; only other *fi*'s can be on the same line with a *fi*. The elision of *else* and *if* into an *elif* is supported. Only a single *fi* is required to close an *if* ... *elif* ... [*else* ...] sequence.

include expression

The expression must evaluate to a file name. The file must contain *bs* source statements. Such statements become part of the program being compiled. *Include* statements may not be nested.

obase *N*

Obase sets the output base to *N* (see *ibase* above).

onintr label**onintr**

The *onintr* command provides program control of interrupts. In the first form, control will pass to the label given, just as if a *goto* had been executed at the time *onintr* was executed. The effect of the statement is cleared after each interrupt. In the second form, an interrupt will cause *bs* to terminate.

return [expression]

The expression is evaluated and the result is passed back as the value of a function call. If no expression is given, zero is returned.

run

The random number generator is reset. Control is passed to the first internal statement. If the *run* statement is contained in a file, it should be the last statement.

stop

Execution of internal statements is stopped. *Bs* reverts to immediate mode.

trace [expression]

The *trace* statement controls function tracing. If the expression is null (or evaluates to zero), tracing is turned off. Otherwise, a record of user-function calls/returns will be printed. Each *return* decrements the *trace* expression value.

while expression statement**while** expression

...

next

While is similar to *for* except that only the conditional expression for loop-continuation is given.

! shell command

An immediate escape to the Shell.

...

This statement is ignored. It is used to interject commentary in a program.

Expression Syntax:**name**

A name is used to specify a variable. Names are composed of a letter (upper or lower case) optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared in *fun* statements, all names are global to the program. Names can take on numeric (double float) values, string values, or can be associated with input/output [see the built-in function *open()* below].

name ([expression [, expression] ...])

Functions can be called by a name followed by the arguments in parentheses separated by commas. Except for built-in functions (listed below), the name must be defined with a *fun* statement. Arguments to functions are passed by value.

name | expression [, expression] ...]

This syntax is used to reference either arrays or tables (see built-in *table* functions below). For arrays, each expression is truncated to an integer and used as a specifier for the name. The resulting array reference is syntactically identical to a name; `a[1,2]` is the same as `a[1][2]`. The truncated expressions are restricted to values between 0 and 32767.

number

A number is used to represent a constant value. A number is written in FORTRAN style, and contains digits, an optional decimal point, and possibly a scale factor consisting of an `e` followed by a possibly signed exponent.

string

Character strings are delimited by `"` characters. The `\` escape character allows the double quote (`\"`), new-line (`\n`), carriage return (`\r`), backspace (`\b`), and tab (`\t`) characters to appear in a string. Otherwise, `\` stands for itself.

(expression)

Parentheses are used to alter the normal order of evaluation.

(expression, expression [, expression ...]) [expression]

The bracketed expression is used as a subscript to select a comma-separated expression from the parenthesized list. List elements are numbered from the left, starting at zero. The expression:

```
( False, True )( a == b )
```

has the value `True` if the comparison is true.

? expression

The interrogation operator tests for the success of the expression rather than its value. At the moment, it is useful for testing end-of-file (see examples in the *Programming Tips* section below), the result of the *eval* built-in function, and for checking the return from user-written functions (see *freturn*). An interrogation "trap" (end-of-file, etc.) causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

- expression

The result is the negation of the expression.

++ name

Increments the value of the variable (or array reference). The result is the new value.

-- name

Decrements the value of the variable. The result is the new value.

! expression

The logical negation of the expression. Watch out for the shell escape command.

expression *operator* expression

Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. Except for the assignment, concatenation, and relational operators, both operands are converted to numeric form before the function is applied.

Binary Operators (in increasing precedence):

`=` is the assignment operator. The left operand must be a name or an array element. The result is the right operand. Assignment binds right to left, all other operators bind left to right.

- (underscore) is the concatenation operator.
- & |
& (logical and) has result zero if either of its arguments are zero. It has result one if both of its arguments are non-zero; | (logical or) has result zero if both of its arguments are zero. It has result one if either of its arguments is non-zero. Both operators treat a null string as a zero.
- < <= > >= == !=
The relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, != not equal to) return one if their arguments are in the specified relation. They return zero otherwise. Relational operators at the same level extend as follows: $a > b > c$ is the same as $a > b$ & $b > c$. A string comparison is made if both operands are strings.
- + -
Add and subtract.
- / %
Multiply, divide, and remainder.
- ^
Exponentiation.

Built-in Functions:

Dealing with arguments

- arg(i)**
is the value of the *i*-th actual parameter on the current level of function call. At level zero, *arg* returns the *i*-th command-line argument (*arg*(0) returns *bs*).
- narg()**
returns the number of arguments passed. At level zero, the command argument count is returned.

Mathematical

- abs(x)**
is the absolute value of *x*.
- atan(x)**
is the arctangent of *x*. Its value is between $-\pi/2$ and $\pi/2$.
- ceil(x)**
returns the smallest integer not less than *x*.
- cos(x)**
is the cosine of *x* (radians).
- exp(x)**
is the exponential function of *x*.
- floor(x)**
returns the largest integer not greater than *x*.
- log(x)**
is the natural logarithm of *x*.

rand()

is a uniformly distributed random number between zero and one.

sin(x)

is the sine of x (radians).

sqrt(x)

is the square root of x .

*String operations***size(s)**

the size (length in bytes) of s is returned.

format(f, a)

returns the formatted value of a . F is assumed to be a format specification in the style of *printf(3S)*. Only the $\%...f$, $\%...e$, and $\%...s$ types are safe.

index(x, y)

returns the number of the first position in x that any of the characters from y matches. No match yields zero.

trans(s, f, t)

Translates characters of the source s from matching characters in f to a character in the same position in t . Source characters that do not appear in f are copied to the result. If the string f is longer than t , source characters that match in the excess portion of f do not appear in the result.

substr(s, start, width)

returns the sub-string of s defined by the *starting* position and *width*.

match(string, pattern)**mstring(n)**

The *pattern* is similar to the regular expression syntax of the *ed(1)* command. The characters \cdot , $[$, $]$, \wedge (inside brackets), \ast and $\$$ are special. The *mstring* function returns the n -th ($1 \leq n \leq 10$) substring of the subject that occurred between pairs of the pattern symbols $\{($ and $\}$ for the most recent call to *match*. To succeed, patterns must match the beginning of the string (as if all patterns began with \wedge). The function returns the number of characters matched. For example:

```
match("a123ab123", "\.[^a-z]\>") == 6
mstring(1) == "b"
```

*File handling***open(name, file, function)****close(name)**

The *name* argument must be a *bs* variable name (passed as a string). For the *open*, the *file* argument may be 1) a 0 (zero), 1, or 2 representing standard input, output, or error output, respectively; 2) a string representing a file name; or 3) a string beginning with an ! representing a command to be executed (via *sh -c*). The *function* argument must be either *r* (read), *w* (write), *W* (write without new-line), or *a* (append). After a *close*, the *name* reverts to being an ordinary variable. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

Examples are given in the following section.

access(s, m)

executes *access(2)*.

ftype(s)

returns a single character file type indication: **f** for regular file, **p** for FIFO (i.e., named pipe), **d** for directory, **b** for block special, or **c** for character special.

*Tables***table(name, size)**

A table in *bs* is an associatively accessed, single-dimension array. "Subscripts" (called keys) are strings (numbers are converted). The *name* argument must be a *bs* variable name (passed as a string). The *size* argument sets the minimum number of elements to be allocated. *Bs* prints an error message and stops on table overflow.

item(name, i)**key()**

The *item* function accesses table elements sequentially (in normal use, there is no orderly progression of key values). Where the *item* function accesses values, the *key* function accesses the "subscript" of the previous *item* call. The *name* argument should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table; for example:

```
table("t", 100)
...
# If word contains "party", the following expression adds one
# to the count of that word:
++t[word]
...
# To print out the the key/value pairs:
for i = 0, ?(s = item(t, i)), ++i if key() put = key()_"_s
```

iskey(name, word)

The *iskey* function tests whether the key *word* exists in the table *name* and returns one for true, zero for false.

*Odds and ends***eval(s)**

The string argument is evaluated as a *bs* expression. The function is handy for converting numeric strings to numeric internal form. *Eval* can also be used as a crude form of indirection, as in:

```
name = "xyz"
eval("++_"_name)
```

which increments the variable *xyz*. In addition, *eval* preceded by the interrogation operator permits the user to control *bs* error conditions. For example:

```
?eval("open(\"X\", \"XXX\", \"r\")")
```

returns the value zero if there is no file named "XXX" (instead of halting the user's program). The following executes a *goto* to the label *L* (if it exists):

```
label="L"
if !(?eval("goto "_label)) puterr == "no label"
```

plot(request, args)

The *plot* function produces output on devices recognized by *tplot(1G)*. The *requests* are as follows:

<i>Call</i>	<i>Function</i>
plot(0, term)	causes further <i>plot</i> output to be piped into <i>tplot</i> (IG) with an argument of <i>-Term</i> .
plot(4)	"erases" the plotter.
plot(2, string)	labels the current point with <i>string</i> .
plot(3, x1, y1, x2, y2)	draws the line between (x1,y1) and (x2,y2).
plot(4, x, y, r)	draws a circle with center (x,y) and radius <i>r</i> .
plot(5, x1, y1, x2, y2, x3, y3)	draws an arc (counterclockwise) with center (x1,y1) and endpoints (x2,y2) and (x3,y3).
plot(6)	is not implemented.
plot(7, x, y)	makes the current point (x,y).
plot(8, x, y)	draws a line from the current point to (x,y).
plot(9, x, y)	draws a point at (x,y).
plot(10, string)	sets the line mode to <i>string</i> .
plot(11, x1, y1, x2, y2)	makes (x1,y1) the lower left corner of the plotting area and (x2,y2) the upper right corner of the plotting area.
plot(12, x1, y1, x2, y2)	causes subsequent x (y) coordinates to be multiplied by x1 (y1) and then added to x2 (y2) before they are plotted. The initial scaling is plot(12, 1.0, 1.0, 0.0, 0.0) .

Some requests do not apply to all plotters. All requests except zero and twelve are implemented by piping characters to *tplot* (IG). See *plot* (4) for more details.

last()

in immediate mode, *last* returns the most recently computed value.

PROGRAMMING TIPS

Using *bs* as a calculator:

```

$ bs
# Distance (inches) light travels in a nanosecond.
186000 * 5280 * 12 / 1e9
11.78496
...

# Compound interest (6% for 5 years on $1,000).
int = .06 / 4
bal = 1000
for i = 1 5*4 bal = bal + bal*int
bal = 1000
346.855007
...
exit

```

The outline of a typical *bs* program:

```
# initialize things:
var1 = 1
open("read", "infile", "r")
...
# compute:
while ?(str = read)
    ...
next
# clean up:
close("read")
...
# last statement executed (exit or stop):
exit
# last input line:
run
```

Input/Output examples:

```
# Copy "oldfile" to "newfile".
open("read", "oldfile", "r")
open("write", "newfile", "w")
...
while ?(write = read)
...
# close "read" and "write":
close("read")
close("write")

# Pipe between commands.
open("ls", "ls *", "r")
open("pr", "!pr -2 -h 'List'", "w")
while ?(pr = ls) ...
...
# be sure to close (wait for) these:
close("ls")
close("pr")
```

SEE ALSO

ed(1), *sh*(1), *tplot*(1G), *access*(2), *printf*(3S), *stdio*(3S), *plot*(4) in the Software Development System manual. See Section 3 of the Software Development System manual for a further description of the mathematical functions [*pow* on *exp*(3M) is used for exponentiation]; *bs* uses the Standard Input/Output package.

NAME

cal - print calendar

SYNOPSIS

cal [[month] year]

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present month is printed. *Year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

BUGS

The year is always considered to start in January even though this is historically naive.

Beware that "cal 83" refers to the early Christian era, not the 20th century.

CALENDAR(1)

NAME

calendar — reminder service

SYNOPSIS

calendar [-]

DESCRIPTION

Calendar consults the file *calendar* in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as "Aug. 24," "august 24," "8/24," etc., are recognized, but not "24 August" or "24/8". On weekends "tomorrow" extends through Monday.

When an argument is present, *calendar* does its job for every user who has a file *calendar* in the login directory and sends them any positive results by *mail*(1). Normally this is done daily by facilities in the UNIX operating system.

FILES

/usr/lib/calprog to figure out today's and tomorrow's dates

/etc/passwd

/tmp/cal*

SEE ALSO

mail(1).

BUGS

Your calendar must be public information for you to get reminder service. *Calendar's* extended idea of "tomorrow" does not account for holidays.

NAME

`cat` - concatenate and print files

SYNOPSIS

```
cat [ -u ] [ -s ] [ -v [-t] [-e] ] file ...
```

DESCRIPTION

Cat reads each *file* in sequence and writes it on the standard output. Thus:

```
cat file
```

prints the file, and:

```
cat file1 file2 >file3
```

concatenates the first two files and places the result on the third.

If no input file is given, or if the argument `-` is encountered, *cat* reads from the standard input file. Output is buffered unless the `-u` option is specified. The `-s` option makes *cat* silent about non-existent files.

The `-v` option causes non-printing characters (with the exception of tabs, new-lines and form-feeds) to be printed visibly. Control characters are printed `^X` (control-*x*); the DEL character (octal 0177) is printed `^?`. Non-ASCII characters (with the high bit set) are printed as `M-x`, where *x* is the character specified by the seven low order bits.

When used with the `-v` option, `-t` causes tabs to be printed as `^I`'s, and `-e` causes a `$` character to be printed at the end of each line (prior to the new-line). The `-t` and `-e` options are ignored if the `-v` option is not specified.

WARNING

Command formats such as

```
cat file1 file2 >file1
```

will cause the original data in *file1* to be lost; therefore, take care when using shell special characters.

SEE ALSO

`cp(1)`, `pg(1)`, `pr(1)`.

CB(1)

NAME

`cb` - C program beautifier

SYNOPSIS

`cb [-s] [-j] [-l leng] [file ...]`

DESCRIPTION

`Cb` reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code. Under default options, `cb` preserves all user newlines. Under the `-s` flag `cb` canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*. The `-j` flag causes split lines to be put back together. The `-l` flag causes `cb` to split lines that are longer than `leng`.

SEE ALSO

`cc(1)`.

The C Programming Language by B. W. Kernighan and D. M. Ritchie.

BUGS

Punctuation that is hidden in preprocessor statements will cause indentation errors.

NAME

cc - C compiler

SYNOPSIS

cc [options] files

DESCRIPTION

The `cc` command is the interface to the C Compilation System. The compilation tools consist of a preprocessor, compiler, optimizer, assembler and link-editor. The `cc` command processes the supplied options and then executes the various tools with the proper arguments. The `cc` command accepts several types of files as arguments:

Files whose names end with `.c` are taken to be C source programs and may be preprocessed, compiled, optimized, assembled and link-edited. The compilation process may be stopped after the completion of any pass if the appropriate options are supplied. If the compilation process runs through the assembler then an object program is produced and is left on the file whose name is that of the source with `.o` substituted for `.c`. However, the `.o` file is normally deleted if a single C program is compiled and link-edited all at one go. In the same way, files whose names end in `.s` are taken to be assembly source programs, and may be assembled and link-edited; and files whose names end in `.i` are taken to be preprocessed C source programs and may be compiled, optimized, assembled and link-edited. Assembly source programs are also run through the C preprocessor before being handed to the assembler. Files whose names do not end in `.c`, `.s` or `.i` are handed to the link editor.

Since the `cc` command usually creates files in the current directory during the compilation process, it is typically necessary to run the `cc` command in a directory in which a file can be created.

The following options are interpreted by `cc`.

- c Suppress the link-editing phase of the compilation, and do not remove any produced object files.
- o *outfile*
Produce an output object file by the name *outfile*. The name of the default file is **a.out**. This is a link-editor option.
- p Arrange for the compiler to produce code that counts the number of times each routine is called; also, if link editing takes place, profiled versions of `libc.a` and `libm.a` (with `-lm` option) are linked and `monitor(3C)` is automatically called. A **mon.out** file will then be produced at normal termination of execution of the object program. An execution profile can then be generated by use of `prof(1)`.
- Bstring
- t/p02a1
These options will be removed in the next release. Use the **-Y** option.
- E Run only `cpp(1)` on the named C programs, and send the result to the standard output.
- O Do compilation phase optimization. This option will not have any affect on `.s` files.
- P Run only `cpp(1)` on the named C programs and and leave the result on corresponding files suffixed `.i`. This option is passed to `cpp(1)`.

- S Compile and do not assemble the named C programs, and leave the assembler-language output on corresponding files suffixed .s.
- q Causes the compiler to generate additional information needed for the use of *sdb(1)*.
- V Print the version of the compiler, optimizer, assembler and/or link-editor that is invoked.
- Wc,arg1[,arg2...]
Hand off the argument[s] *argi* to pass *c* where *c* is one of [p02a] indicating the preprocessor, compiler, optimizer, assembler, or link editor, respectively. For example: -Wa,-m passes -m to the assembler.
- Y [p02a]bSILU, *dirname*
Specify a new pathname, *dirname*, for the locations of the tools and directories designated in the first argument. [p02a]bSILU represents:
 - p preprocessor
 - 0 compiler
 - 2 optimizer
 - a assembler
 - l link editor
 - b basicblk (used by -q1 and -qx)
 - S directory containing the start-up routines
 - I default include directory searched by *cpp(1)*
 - L first default library directory searched by *ld(1)*
 - U second default library directory searched by *ld(1)*

If the location of a tool is being specified, then the new pathname for the tool will be <*dirname*>/<*tool*>. If more than one -Y option is applied to any one tool or directory, then the last occurrence holds.
- # The -# options provide information about the execution of cc. The -# option prints, for each tool called, the name of the tool followed by each option which is passed to the tool. The -## option prints the output of the -# option followed by the absolute pathname of the tool. The -### option prints the same output as the -## option but does not exec the tool. The output format of these options is not guaranteed from one release to the next though the content will be the same.
- Ms,-Ml
Generate small memory model code (16 bit addressing) and generate large memory model code (32 bit addressing), respectively. Small model is the default. Small and large model outputs are incompatible. Refer to the SW Dev. Guide - Programming on the S5.

The cc command also recognizes -C, -D, -H, -I and -U and passes these options and their arguments directly to the preprocessor without using the -W option. Similarly, the cc command recognizes -a, -k, -l, -m, -o, -r, -s, -t, -u, -x, -z, -K, -L, and -V and passes these options and their arguments directly to the loader. See the manual pages for *cpp(1)* and *ld(1)* for descriptions.

Other arguments are taken to be C-compatible object programs, typically produced by an earlier cc run, or perhaps libraries of C-compatible routines and are passed directly to the link editor. These programs, together with the results of any compilations specified, are link-edited (in the

order given) to produce an executable program with name **a.out** unless the **-o** option of the link-editor is used.

If the **cc** command is put in a file *prefixcc* the prefix will be parsed off the command and used to call the tools, i.e., *prefixtool*. For example, **OLDcc** will call **OLDcpp**, **OLDcomp**, **OLDoptim**, **OLDbasicblk**, **OLDas**, and **OLDld** and will link **OLDcrt0.o**. Therefore, one **MUST** be careful when moving the **cc** command around. The prefix will apply to the preprocessor, compiler, optimizer, assembler, link-editor and the start-up routines.

The C language standard was extended to allow arbitrary length variable names. The option pair "**-Wp,-T-WO,-XT**" will cause **cc** to truncate arbitrary length variable names.

FILES

file.c	C source file
file.i	preprocessed C source file
file.o	object file
file.s	assembly language file
a.out	link-edited output
crt0.o	start-up routine
TEMPDIR/*	temporary
LIBDIR/	basic analyzer
LIBDIR/cpp	preprocessor, <i>cpp</i> (1)
LIBDIR/comp	compiler
LIBDIR/optim	optimizer
BINDIR/as	assembler, <i>as</i> (1)
BINDIR/ld	link editor, <i>ld</i> (1)
LIBDIR/libc.a	library

LIBDIR is usually **/lib**; **BINDIR** is usually **/bin**; and **TEMPDIR** is usually **/usr/tmp**.

SEE ALSO

Kernighan, B. W., and Ritchie, D. M., *The C Programming Language*, Prentice-Hall, 1978.

Kernighan, B. W., *Programming in C-A Tutorial*.

Ritchie, D. M., *C Reference Manual*.

as(1), *ld*(1), *cpp*(1), *lint*(1), *prof*(1), *sdb*(1).

DIAGNOSTICS

The diagnostics produced by the C compiler are intended to be self-explanatory. Occasional messages may be produced by the assembler or link-editor.

NOTES

By default, the return value from a C program is completely random. The only two guaranteed ways to return a specific value is to explicitly call *exit*(2) or to leave the function *main*() with a "return expression;" construct.

CD(1)

NAME

cd -- change working directory

SYNOPSIS

cd [*directory*]

DESCRIPTION

If *directory* is not specified, the value of shell parameter *\$HOME* is used as the new working directory. If *directory* specifies a complete path starting with */*, *..*, *..*, *directory* becomes the new working directory. If neither case applies, *cd* tries to find the designated directory relative to one of the paths specified by the *\$CDPATH* shell variable. *\$CDPATH* has the same syntax as, and similar semantics to, the *\$PATH* shell variable. *cd* must have execute (search) permission in *directory*.

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command; therefore, it is recognized and is internal to the shell.

SEE ALSO

pwd(1), *sh*(1).

chdir(2) in the Software Development System manual.

NAME

`cdc` — change the delta commentary of an SCCS delta

SYNOPSIS

`cdc -rSID [-m[mrlist]] [-y[comment]] files`

DESCRIPTION

`Cdc` changes the *delta commentary*, for the *SID* specified by the `-r` keyletter, of each named SCCS file.

Delta commentary is defined to be the Modification Request (MR) and comment information normally specified via the *delta(1)* command (`-m` and `-y` keyletters).

If a directory is named, `cdc` behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with `s.`) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read [see *WARNINGS*]; each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to `cdc`, which may appear in any order, consist of *keyletter* arguments and file names.

All the described *keyletter* arguments apply independently to each named file:

`-rSID` Used to specify the SCCS IDentification (*SID*) string of a delta for which the delta commentary is to be changed.

`-m[mrlist]` If the SCCS file has the `v` flag set [see *admin(1)*] then a list of MR numbers to be added and/or deleted in the delta commentary of the *SID* specified by the `-r` keyletter may be supplied. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of *delta(1)*. In order to delete an MR, precede the MR number with the character `!` (see *EXAMPLES*). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a "comment" line. A list of all deleted MRs is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If `-m` is not used and the standard input is a terminal, the prompt `MRs?` is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The `MRs?` prompt always precedes the `comments?` prompt (see `-y` keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the `v` flag has a value [see *admin(1)*], it is taken to be the name of a program (or shell procedure) which validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, `cdc` terminates and the delta commentary remains unchanged.

`-y[comment]` Arbitrary text used to replace the *comment(s)* already existing for the delta specified by the `-r` keyletter. The previous comments are kept and preceded by a comment

CDC(1)

line stating that they were changed. A null *comment* has no effect.

If `-y` is not specified and the standard input is a terminal, the prompt `comments?` is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the *comment* text.

The exact permissions necessary to modify the SCCS file are documented in the *Source Code Control System User Guide*. Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLES

```
cdc -r1.6 -m"b178-12345 !b177-54321 b179-00001" -ytrouble s.file
```

adds b178-12345 and b179-00001 to the MR list, removes b177-54321 from the MR list, and adds the comment `trouble` to delta 1.6 of s.file.

```
cdc -r1.6 s.file
```

```
MRs? !b177-54321 b178-12345 b179-00001  
comments? trouble
```

does the same thing.

WARNINGS

If SCCS file names are supplied to the `cdc` command via the standard input (`--` on the command line), then the `-m` and `-y` keyletters must also be used.

FILES

x-file [see *delta(1)*]

z-file [see *delta(1)*]

SEE ALSO

admin(1), *delta(1)*, *get(1)*, *help(1)*, *prs(1)*.

scsfile(4) in the Software Development System manual.

"Source Code Control System User Guide" in the Software Development System manual.

DIAGNOSTICS

Use *help(1)* for explanations.

NAME

`cflow` - generate C flow graph

SYNOPSIS

`cflow` [-r] [-ix] [-i_] [-dnum] files

DESCRIPTION

Cflow analyzes a collection of C, YACC, LEX, assembler, and object files and attempts to build a graph charting the external references. Files suffixed in `.y`, `.l`, `.c`, and `.i` are YACC'd, LEX'd, and C-preprocessed (bypassed for `.i` files) as appropriate and then run through the first pass of *lint*(1). (The `-I`, `-D`, and `-U` options of the C-preprocessor are also understood.) Files suffixed with `.s` are assembled and information is extracted (as in `.o` files) from the symbol table. The output of all this nontrivial processing is collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference (i.e., line) number, followed by a suitable number of tabs indicating the level. Then the name of the global (normally only a function not defined as an external or beginning with an underscore; see below for the `-i` inclusion option) a colon and its definition. For information extracted from C source, the definition consists of an abstract type declaration (e.g., `char *`), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (e.g., `text`). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only `<>` is printed.

As an example, given the following in *file.c*:

```
int    i;

main()
{
    f();
    g();
    f();
}

f()
{
    i = h();
}
```

the command

```
cflow -ix file.c
```

produces the output

```
1      main: int(), <file.c 4>
2      f: int(), <file.c 11>
3      h: <>
4      i: int, <file.c 1>
5      g: <>
```

When the nesting level becomes too deep, the `-e` option of *pr*(1) can be used

CFLOW(1)

to compress the tab expansion to something less than every eight spaces.

The following options are interpreted by *cflow*:

- r Reverse the "caller: callee" relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.
- ix Include external and static data symbols. The default is to include only functions in the flowgraph.
- i_ Include names that begin with an underscore. The default is to exclude these functions (and data if *-ix* is used).
- dnum The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be met with contempt.

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C-preprocessor).

SEE ALSO

as(1), cc(1), cpp(1), lex(1), lint(1), nm(1), pr(1), yacc(1).

BUGS

Files produced by *lex*(1) and *yacc*(1) cause the reordering of line number declarations which can confuse *cflow*. To get proper results, feed *cflow* the *yacc* or *lex* input.

NAME

checkall - faster file system checking procedure

SYNOPSIS

/etc/checkall

DESCRIPTION

The *checkall* procedure is a prototype and must be modified to suit local conditions. The following will serve as an example:

```
# check the root file system by itself
fsck /dev/dsk/0s0
```

```
# dual fsck of drives 0 and 1
dfsk /dev/rdisk/0s[12345] - /dev/rdisk/1s1
```

In the above example (where */dev/rdisk/1s1* is 320K blocks and */dev/rdisk/0s[12345]* are each 65K or less), a previous sequential *fsck* took 19 minutes. The *checkall* procedure takes 11 minutes.

Dfsck is a program that permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsk* will print the file system name for each message to the operator. When answering a question from *dfsk*, the operator must prefix the response with a 1 or a 2 (indicating that the answer refers to the first or second file system group).

Due to the file system load balancing required for dual checking, the *dfsk* command should always be executed through the *checkall* shell procedure.

In a practical sense, the file systems are divided as follows:

```
dfsk file_systems_on_drive_0 - file_systems_on_drive_1
dfsk file_systems_on_drive_2 - file_systems_on_drive_3
```

...

A three-drive system can be handled by this more concrete example (assumes two large file systems per drive):

```
dfsk /dev/dsk/3s1 /dev/dsk/0s[14] - /dev/dsk/1s[14] /dev/dsk/3s4
```

Note that the first file system on drive 3 is first in the *fileystems1* list and is last in the *fileystems2* list assuring that references to that drive will not overlap at execution time.

CHECKALL(1M)

WARNINGS

1. Do not use *dfsk* to check the *root* file system.
2. On a check that requires a scratch file (see *-t* above), be careful not to use the same temporary file for the two groups (this is sure to scramble the file systems).
3. The *dfsk* procedure is useful only if the system is set up for multiple physical I/O buffers.

SEE ALSO

fsck(1M).
"Single User and Multiuser" in this manual.

NAME

chmod — change mode

SYNOPSIS

chmod mode files

DESCRIPTION

The permissions of the named *files* are changed according to *mode*, which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, see <i>chmod(2)</i>
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*]

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, the default if *who* is omitted.

Op can be **+** to add *permission* to the file's mode, **-** to take away *permission*, or **=** to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group ID) and **t** (save text, or sticky); **u**, **g**, or **o** indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with **=** to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter **s** is only useful with **u** or **g** and **t** only works with **u**.

Only the owner of a file (or the super-user) may change its mode. Only the super-user may set the sticky bit. In order to set the group ID, the group of the file must correspond to your current group ID.

EXAMPLES

The first example denies write permission to others, the second makes a file executable:

```
chmod o-w file
```

```
chmod +x file
```

SEE ALSO

ls(1).

chmod(2) in the Software Development System manual.

CHOWN(1)

NAME

chown, **chgrp** — change owner or group

SYNOPSIS

chown owner file ...

chgrp group file ...

DESCRIPTION

Chown changes the owner of the *files* to *owner*. The owner may be either a decimal user ID or a login name found in the password file.

Chgrp changes the group ID of the *files* to *group*. The group may be either a decimal group ID or a group name found in the group file.

If either command is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

FILES

/etc/passwd

/etc/group

SEE ALSO

chmod(1).

chown(2), group(4), passwd(4) in the Software Development System manual.

NAME
chroot — change root directory for a command

SYNOPSIS
/etc/chroot newroot command

DESCRIPTION
The given command is executed *relative to the new root*. The meaning of any initial slashes (/) in path names is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Notice that:

chroot newroot command >x

will create the file x relative to the original root, not the new one.

This command is restricted to the super-user.

The new root path name is always relative to the current root: even if a *chroot* is currently in effect, the *newroot* argument is relative to the current root of the running process.

SEE ALSO
chdir(2).

BUGS
One should exercise extreme caution when referencing special files in the new root file system.

NAME

clri - clear i-node

SYNOPSIS

/etc/clri *file-system* *i-number* ...

DESCRIPTION

Clri writes zeros on the 64 bytes occupied by the i-node numbered *i-number*. *File-system* must be a special file name referring to a device containing a file system. After *clri* is executed, any blocks in the affected file will show up as "missing" in an *fsck*(1M) of the *file-system*. This command should only be used in emergencies and extreme care should be exercised.

Read and write permission is required on the specified *file-system* device. The i-node becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to *zap* an i-node which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated i-node, so the whole cycle is likely to be repeated again and again.

SEE ALSO

fsck(1M), *fsdb*(1M), *ncheck*(1M), *fs*(4).

BUGS

If the file is open, *clri* is likely to be ineffective.

NAME

`cmp` — compare two files

SYNOPSIS

`cmp [-l] [-s] file1 file2`

DESCRIPTION

The two files are compared. (If *file1* is `-`, the standard input is used.) Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

Options:

- `-l` Print the byte number (decimal) and the differing bytes (octal) for each difference.
- `-s` Print nothing for differing files; return codes only.

SEE ALSO

`comm(1)`, `diff(1)`.

DIAGNOSTICS

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

COL(1)

NAME

`col` - filter reverse line-feeds

SYNOPSIS

`col [-bfpx]`

DESCRIPTION

`col` reads from the standard input and writes onto the standard output. It performs the line overlays implied by reverse line feeds (ASCII code ESC-7), and by forward and reverse half-line feeds (ESC-9 and ESC-8). `col` is particularly useful for filtering multicolumn output made with the `.rt` command of `nroff` and output resulting from use of the `tbl(1)` preprocessor.

If the `-b` option is given, `col` assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, only the last one read will be output.

Although `col` accepts half-line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. This treatment can be suppressed by the `-f` (fine) option; in this case, the output from `col` may contain forward half-line feeds (ESC-9), but will still never contain either kind of reverse line motion.

Unless the `-x` option is given, `col` will convert white space to tabs on output wherever possible to shorten printing time.

The ASCII control characters SO (\016) and SI (\017) are assumed by `col` to start and end text in an alternate character set. The character set to which each input character belongs is remembered, and on output SI and SO characters are generated as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab, return, new-line, SI, SO, VT (\013), and ESC followed by 7, 8, or 9. The VT character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

Normally, `col` will ignore any unknown-to-it escape sequences found in its input; the `-p` option may be used to cause `col` to output these sequences as regular characters, subject to overprinting from reverse line motions. The use of this option is highly discouraged unless the user is fully aware of the textual position of the escape sequences.

SEE ALSO

"Nroff and Troff User manual", "Mathematics Typesetting Program" (`eqn`), and "Table Formatting Program" (`tbl`) in the Text Preparation System manual.

NOTES

The input format accepted by `col` matches the output produced by `nroff` with either the `-T37` or `-Tlp` options. Use `-T37` (and the `-f` option of `col`) if the ultimate disposition of the output of `col` will be a device that can interpret half-line motions, and `-Tlp` otherwise.

BUGS

Cannot back up more than 128 lines.

Allows at most 800 characters, including backspaces, on a line.

Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

NAME

comb — combine SCCS deltas

SYNOPSIS

comb [-o] [-s] [-psid] [-clist] files

DESCRIPTION

Comb generates a shell procedure [see *sh*(1)] which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s,) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored. The generated shell procedure is written on the standard output.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

-psid The SCCS IDentification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

-clist A list [see *get*(1) for the syntax of a list] of deltas to be preserved. All other deltas are discarded.

-o For each *get -e* generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the -o keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

-s This argument causes *comb* to generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

$$100 * (\text{original} - \text{combined}) / \text{original}$$

It is recommended that before any SCCS files are actually combined, one should use this option to determine exactly how much space is saved by the combining process.

If no keyletter arguments are specified, *comb* will preserve only leaf deltas and the minimal number of ancestors needed to preserve the tree.

FILES

s.COMB The name of the reconstructed SCCS file.
comb????? Temporary.

SEE ALSO

admin(1), *delta*(1), *get*(1), *help*(1), *prs*(1), *sh*(1).
sccsfile(4) and "Source Code Control System User Guide" in the Software Development System manual.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

COMM(1)

NAME

`comm` - select or reject lines common to two sorted files

SYNOPSIS

`comm` [- [123]] file1 file2

DESCRIPTION

Comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort*(1)), and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name - means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus `comm -12` prints only the lines common to the two files; `comm -23` prints only lines in the first file but not in the second; `comm -123` is a no-op.

SEE ALSO

`cmp`(1), `diff`(1), `sort`(1), `uniq`(1).

NAME

config — configure a UNIX system

SYNOPSIS

/etc/config [-t] [-l file] [-c file] [-m file] dfile

DESCRIPTION

Config is a program that takes a description of a UNIX system and generates two files. One file provides information regarding the interface between the hardware and device handlers. The other file is a C program defining the configuration tables for the various devices on the system.

The **-l** option specifies the name of the hardware interface file; **handlers.c** is the default name.

The **-c** option specifies the name of the configuration table file; **conf.c** is the default name.

The **-m** option specifies the name of the file that contains all the information regarding supported devices; **/etc/master** is the default name. This file is supplied with the UNIX system and should *not* be modified unless the user *fully* understands its construction.

The **-t** option requests a short table of major device numbers for character and block type devices. This can facilitate the creation of special files.

The user must supply *dfile*; it must contain device information for the user's system. This file is divided into two parts. The first part contains physical device specifications. The second part contains system-dependent information. Any line with an asterisk (*) in column 1 is a comment.

First Part of *dfile*

Each line contains up to 3 fields, delimited by blanks and/or tabs in the following format:

```
devname vector number
```

where *devname* is the name of the device (as it appears in the **/etc/master** device table), *vector* is the interrupt vector number (decimal) in the range 32 to 255 or a zero to specify that a default value should be supplied from the **/etc/master** file, *number* is the number (decimal) of devices associated with the corresponding controller. *Number* is optional, and if omitted, a default value which is the maximum value for that controller is used.

There are certain drivers that may be provided with the system, that are actually *pseudo-device* drivers; that is, there is no real hardware associated with the driver. Drivers of this type are identified on their respective manual entries. When these devices are specified in the description file, the interrupt *vector* must be zero.

Second Part of *dfile*

The second part contains three different types of lines. Note that *all* specifications of this part *are required*, although their order is arbitrary.

1. Root/pipe/dump device specification

Three lines of three fields each:

```
root devname minor
pipe devname minor
dump devname minor
```

where *minor* is the minor device number (in decimal).

2. Swap device specification

CONFIG(1M)

One line that contains five fields as follows:

```
swap devname minor swplo nswap
```

where *swplo* is the lowest 512-byte disk block (decimal) in the swap area and *nswap* is the number of disk blocks (decimal) in the swap area.

3. Parameter specification

Several lines of two fields each as follows (*number* is decimal):

```
buffers      number
inodes       number
files        number
mounts       number
coremap      number
swapmap      number
calls        number
procs        number
maxproc      number
texts        number
clists       number
hashbuf      number
physbuf      number
mesg         0 or 1
sema         0 or 1
shmем        0 or 1
```

EXAMPLE

To configure an Intel System 286/310 with one disk drive controller with one 16-Mb Winchester disk, and one floppy drive, the following parameter information must be specified:

```
root device is a Winchester(drive 0, section 6)
pipe device is a Winchester(drive 0, section 6)
swap device is a Winchester(drive 0, section 6),
    with a swplo of 8192 and an nswap of 4096
dump device is a floppy(drive 0, section 24),
number of buffers is 63
number of processes is 75
maximum number of processes per user ID is 25
number of mounts is 8
number of inodes is 75
number of files is 75
number of calls is 50
number of texts is 40
number of character buffers is 150
number of coremap entries is 75
number of swapmap entries is 75
shared memory is to be included
messages are to be included
semaphores are to be included
```

The actual system configuration would be specified as follows:

```
wini         0          1
flp          0          1
root         wini       6
pipe         wini       6
swap         wini       6          8192    4096
dump         flp        24
• Comments may be inserted in this manner
buffers      63
```


procs	75
maxproc	25
mounts	8
inodes	75
files	75
calls	50
texts	40
clists	150
coremap	75

swapmap	75
shmem	1
msg	1
sema	1

FILES

/etc/master	default input master device table
handlers.c	default output hardware interface file
conf.c	default output configuration table file

SEE ALSO

sysdef(1M), master(4).

DIAGNOSTICS

Diagnostics are routed to the standard output and are self-explanatory.

BUGS

The `-t` option does not know about devices that have aliases. For example, a `dsk` (an alias for a `wini`) will show up as a `wini`; however, the major device numbers are always correct.

CP(1)

NAME

cp, *ln*, *mv* — copy, link or move files

SYNOPSIS

```
cp file1 [ file2 ...] target  
ln [ -f ] file1 [ file2 ...] target  
mv [ -f ] file1 [ file2 ...] target
```

DESCRIPTION

File1 is copied (linked, moved) to *target*. Under no circumstance can *file1* and *target* be the same [take care when using *sh*(1) metacharacters]. If *target* is a directory, then one or more files are copied (linked, moved) to that directory. If *target* is a file, its contents are destroyed.

If *mv* or *ln* determines that the mode of *target* forbids writing, it will print the mode [see *chmod*(2)], ask for a response, and read the standard input for one line; if the line begins with *y*, the *mv* or *ln* occurs, if permissible; if not, the command exits. No questions are asked and the *mv* or *ln* is done when the *-f* option is used or if the standard input is not a terminal.

Only *mv* will allow *file1* to be a directory, in which case the directory rename will occur only if the two directories have the same parent; *file1* is renamed *target*. If *file1* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file.

When using *cp*, if *target* is not a file, a new file is created which has the same mode as *file1* except that the sticky bit is not set unless you are super-user; the owner and group of *target* are those of the user. If *target* is a file, copying a file into *target* does not change its mode, owner, nor group. The last modification time of *target* (and last access time, if *target* did not exist) and the last access time of *file1* are set to the time the copy was made. If *target* is a link to a file, all links remain and the file is changed.

SEE ALSO

cpio(1), *ln*(1),
chmod(2) in the Software Development System manual.

BUGS

If *file1* and *target* lie on different file systems, *mv* must copy the file and delete the original. In this case any linking relationship with other files is lost.

Ln will not link across file systems.

NAME

cpio - copy file archives in and out

SYNOPSIS

cpio -o [**acBv**]

cpio -i [**BcdmrtuvfsSb6**] [**patterns**]

cpio -p [**adlmruv**] **directory**

DESCRIPTION

cpio -o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. Output is padded to a 512-byte boundary.

cpio -i (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio -o**. Only files with names that match *patterns* are selected. *Patterns* are given in the name-generating notation of *sh(1)*. In *patterns*, meta-characters **?**, *****, and **[...]** match the slash **/** character. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is ***** (i.e., select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files will be those of the previous **cpio -o**. The owner and group of the files will be that of the current user unless the user is super-user, which causes *cpio* to retain the owner and group of the files of the previous **cpio -o**.

cpio -p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

The meanings of the available options are:

- a** Reset access times of input files after they have been copied.
- B** Input/output is to be blocked 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from **/dev/rmt/??**).
- d** *Directories* are to be created as needed.
- c** Write *header* information in ASCII character form for portability.
- r** Interactively *rename* files. If the user types a null line, the file is skipped.
- t** Print a *table of contents* of the input. No files are created.
- u** Copy *unconditionally* (normally, an older file will not replace a newer file with the same name).
- v** *Verbose*: causes a list of file names to be printed. When used with the **t** option, the table of contents looks like the output of an **ls -l** command [see *ls(1)*].
- l** Whenever possible, link files rather than copying them. Usable only with the **-p** option.
- m** Retain previous file modification time. This option is ineffective on directories that are being copied.
- f** Copy in all files except those in *patterns*.
- s** Swap bytes. Use only with the **-i** option.
- S** Swap halfwords. Use only with the **-i** option.
- b** Swap both bytes and halfwords. Use only with the **-i** option.
- 6** Process an old (i.e., UNIX System *Sixth* Edition format) file. Only useful with **-i** (copy in).

EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

CPIO(1)

```
ls | cpio -o >/dev/mt/0m  
cd olddir  
find . -depth -print | cpio -pdl newdir
```

The trivial case "find . -depth -print | cpio -oB >/dev/rmt/0m" can be handled more efficiently by:

```
find . -cpio /dev/rmt/0m
```

SEE ALSO

ar(1), find(1), ls(1).

cpio(4) in the Software Development System manual.

BUGS

Path names are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost. Only the super-user can copy special files. The -B option does not work with certain magnetic tape drives.

NAME

cpp - the C language preprocessor

SYNOPSIS

`/lib/cpp [option ...] [ifile [ofile]]`

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc(1)* command. Thus the output of *cpp* is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, *cpp* and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of *cpp* other than in this framework is not suggested. The preferred way to invoke *cpp* is through the *cc(1)* command since the functionality of *cpp* may someday be moved elsewhere. See *m4(1)* for a general macro processor.

Cpp optionally accepts two file names as arguments. *Ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following *options* to *cpp* are recognized:

- P Preprocess the input without producing the line control information used by the next pass of the C compiler.
- C By default, *cpp* strips C-style comments. If the *-C* option is specified, all comments (except those found on *cpp* directive lines) are passed along.
- U*name*
Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor.
- D*name*
-D*name*=*def*
Define *name* as if by a *#define* directive. If no *=def* is given, *name* is defined as 1. The *-D* option has lower precedence than the *-U* option. That is, if the same name is used in both a *-U* option and a *-D* option, the name will be undefined regardless of the order of the options.
- I*dir* Change the algorithm for searching for *#include* files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, *#include* files whose names are enclosed in " " will be searched for first in the directory of the *ifile* argument, then in directories named in *-I* options, and last in directories on a standard list. For *#include* files whose names are enclosed in <>, the directory of the *ifile* argument is not searched.
- H Print, one per line on standard error, the full path names of included files.
- T Preprocessor symbols are no longer restricted to eight characters. The *-T* option forces *cpp* to use only the first eight characters for distinguishing different preprocessor names. This behavior is the same as previous preprocessors with respect to the length of names and is included for backwards compatibility.

Two special names are understood by *cpp*. The name `__LINE__` is defined as the current line number (as a decimal integer) as known by *cpp*, and `__FILE__` is defined as the current file name (as a C string) as known by *cpp*. They can be used anywhere (including in macros) just as any other defined name.

All *cpp* directives start with lines whose first character is #. Any number of blanks and tabs are allowed between the # and the directive. The directives are:

#define *name token-string*

Replace subsequent instances of *name* with *token-string*.

#define *name(arg, ..., arg) token-string*

Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma-separated tokens, and a) by *token-string* where each occurrence of an *arg* in the *token-string* is replaced by the corresponding set of tokens in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, *cpp* re-starts its scan for names to expand at the beginning of the newly created *token-string*.

#undef *name*

Cause the definition of *name* (if any) to be forgotten from now on.

#include "*filename*"

#include <*filename*>

Include at this point the contents of *filename* (which will then be run through *cpp*). When the <*filename*> notation is used, *filename* is only searched for in the standard places. See the -I option above for more detail.

#line *integer-constant* "*filename*"

Causes *cpp* to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file where it comes from. If "*filename*" is not given, the current file name is unchanged.

#endif

Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

#ifdef *name*

The lines following will appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#ifndef *name*

The lines following will not appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#if *constant-expression*

Lines following will appear in the output if and only if the *constant-expression* evaluates to nonzero. All binary nonassignment C operators, the ? operator, the unary -, !, and ~ operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *cpp* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

#elif *constant-expression*

An arbitrary number of **#elif** directives are allowed between a **#if**

(or `#ifdef` or `#ifndef`) directive and a `#else` or `#endif` directive. The lines following the `#elif` directive will appear in the output if and only if the preceding test directive evaluated to zero, all intervening `#elif` directives evaluated to zero, and if the *constant-expression* evaluates to nonzero. If *constant-expression* evaluates to nonzero, all succeeding `#elif` and `#else` directives will be ignored. Any *constant-expression* allowed in an `#if` directive is allowed in a `#elif` directive.

#else The lines following will appear in the output if and only if all of the previous `#if`, `#ifdef`, `#ifndef`, and `#elif` directives evaluated to zero.

The test directives and the possible `#else` directives can be nested.

FILES

`/usr/include` standard directory for `#include` files

SEE ALSO

`cc(1)`, `m4(1)`.

DIAGNOSTICS

The error messages produced by `cpp` are intended to be self-explanatory. The line number and file name where the error occurred are printed along with the diagnostic.

NOTES

When new-line characters were found in argument lists for macros to be expanded, previous versions of `cpp` put out the new-lines as they were found and expanded. The current version of `cpp` replaces these new-lines with blanks to alleviate problems that the previous versions had when this occurred.

CPSET(1M)

NAME

`cpset` — install object files in binary directories

SYNOPSIS

`cpset [-o] object directory [mode owner group]`

DESCRIPTION

Cpset is used to install the specified *object* file in the given *directory*. The *mode*, *owner*, and *group*, of the destination file may be specified on the command line. If this data is omitted, two results are possible:

If the user of *cpset* has administrative permissions (that is, the user's numerical ID is less than 100), the following defaults are provided:

mode — 0755

owner — bin

group — bin

If the user is not an administrator, the default, owner, and group of the destination file will be that of the invoker.

An optional argument of `--o` will force *cpset* to move *object* to *OLDObject* in the destination directory before installing the new object.

For example:

```
cpset echo /bin 0755 bin bin
```

```
cpset echo /bin
```

```
cpset echo /bin/echo
```

All the examples above have the same effect (assuming the user is an administrator). The file **echo** will be copied into **/bin** and will be given **0755**, **bin**, **bin** as the mode, owner, and group, respectively.

Cpset utilizes the file `/usr/src/destinations` to determine the final destination of a file. The locations file contains pairs of path names separated by spaces or tabs. The first name is the "official" destination (for example: `/bin/echo`). The second name is the new destination. For example, if *echo* is moved from `/bin` to `/usr/bin`, the entry in `/usr/src/destinations` would be:

```
/bin/echo      /usr/bin/echo
```

When the actual installation happens, *cpset* verifies that the "old" path name does not exist. If a file exists at that location, *cpset* issues a warning and continues. This file does not exist on a distribution disk; it is used by sites to track local command movement. The procedures used to build the source will be responsible for defining the "official" locations of the source.

Cross Generation

The environment variable `ROOT` will be used to locate the destination file (in the form `$ROOT/usr/src/destinations`). This is necessary in the cases where cross generation is being done on a production system.

SEE ALSO

`install(1M)`, `make(1)`, `mk(8)`.

NAME

crash - examine system images

SYNOPSIS

/etc/crash [*system*] [*namelist*]

DESCRIPTION

Crash is an interactive utility for examining an operating system core image. It has facilities for interpreting and formatting the various control structures in the system and certain miscellaneous functions that are useful when perusing a dump.

The arguments to *crash* are the file name where the *system* image can be found and a *namelist* file to be used for symbol values.

The default values are **/dev/mem** and **/unix**; hence, *crash* with no arguments can be used to examine an active system. If a *system* image file is given, it is assumed to be a system core dump and the default process is set to be that of the process running at the time of the crash. This is determined by a value stored in a fixed location by the dump mechanism.

COMMANDS

Input to *crash* is typically of the form:

command [*options*] [structures to be printed].

When allowed, *options* will modify the format of the printout. If no specific structure elements are specified, all valid entries will be used. As an example, **proc - 12 15 3** would print process table slots 12, 15, and 3 in a long format, while **proc** would print the entire process table in standard format.

In general, those commands that perform I/O with addresses assume hexadecimal on 32-bit machines and octal on 16-bit machines.

The current repertory consists of:

user [list of process table entries]

Aliases: **uarea, u_area, u.**

Print the user structure of the named process as determined by the information contained in the process table entry. If no entry number is given, the information from the last executing process will be printed. Swapped processes produce an error message.

trace [**-r**] [list of process table entries]

Aliases: **t.**

Generate a kernel stack trace of the current process. If the **-r** option is used, the trace begins at the saved stack frame pointer in **kfp**. Otherwise the trace starts at the bottom of the stack and attempts to find valid stack frames deeper in the stack. If no entry number is given, the information from the last executing process will be printed.

kfp [stack frame pointer]

Aliases: **fp**

Print the program's idea of the start of the current stack frame (set initially from a fixed location in the dump) if no argument is given or set the frame pointer to the supplied value.

stack [list of process table entries]

Aliases: **stk, s, kernel, k.**

Format a dump of the kernel stack of a process. The addresses shown are virtual system data addresses rather than true physical locations. If no entry number is given, the information from the last executing process will be printed.

- proc** [**-r**] [list of process table entries]
 Aliases: **ps**, **p**.
 Format the process table. The **-r** option causes only runnable processes to be printed. The **-** alone generates a longer listing.
- pcb** [list of process table entries]
 Print the process control block of the current process. If no entry number is given, the information from the last executing process will be printed.
- gdt** [selector in hex] [number in decimal]
 Print Global Descriptor Table. If no parameters are specified, then the expanded form of the Global Descriptor Table is printed. All non-zero segment entries are printed. If the selector parameter is given, the non-zero segments are expanded and printed from this point until either the end of the table is reached or the number of segments specified is reached, whichever comes first. (UNIX System V/286 only.)
- ldt** [process number in decimal] [selector in hex] [number in decimal]
 Print Local Descriptor Table. If no parameters are specified, then the current Local Descriptor Table is expanded and printed. All non-zero segment entries are expanded. If the process number is specified, then the Local Descriptor Table for that process is printed. Other parameters are treated in the same manner as in the **gdt** function. (UNIX System V/286 only.)
- idt** [selector in hex] [number in decimal]
 Print Interrupt Descriptor Table. If no parameters are specified, then the expanded form of the Interrupt Descriptor Table is printed. All non-zero segment entries are printed. If the selector parameter is given, the non-zero segments are expanded and printed from this point until either the end of the table is reached or the number of segments specified is reached, whichever comes first. (UNIX System V/286 only.)
- stkbase** [process number in decimal]
 Print bottom of kernel stack. This function allows the user to format the bottom of the kernel stack. The area printed is the interface from the user to kernel by means of a system call or an interrupt while in user mode. If a process number is specified, the kernel stack interface for the process is printed. If no process number is specified, then the current interface is printed. (UNIX System V/286 only.)
- i-node** [**-**] [list of i-node table entries]
 Aliases: **ino**, **i**.
 Format the i-node table. The **-** option will also print the i-node data block addresses.
- file** [list of file table entries]
 Aliases: **files**, **f**.
 Format the file table.
- lck** Aliases: **l**
 Print the active and sleep record lock tables; also verify the correctness of the record locking linked lists.
- mount** [list of mount table entries]
 Aliases: **mnt**, **m**.

- Format the mount table.
- text** [list of text table entries]
Aliases: **txt**, **x**.
Format the text table.
- tty** [*type*] [-] [list of tty entries]
Aliases: **term** (also **sio** and **con** are aliases on Intel machines).
Print the tty structures. The *type* argument determines which structure will be used (such as **sio**, **con**, on Intel equipment). No default *type* is provided. However, once specified, the last *type* is remembered. The - option prints the *stty*(1) parameters for the given line.
- stat** Print certain statistics found in the dump. These include the panic string (if a panic occurred), time of crash, system name, and the registers saved in low memory by the dump mechanism.
- var** Aliases: **tunables**, **tunable**, **tune**, **v**.
Print the tunable system parameters.
- buf** [list of buffer headers]
Aliases: **hdr**, **bufhdr**.
Format the system buffer headers.
- buffer** [format] [list of buffers]
Alias: **b**.
Print the data in a system buffer according to *format*. If *format* is omitted, the previous *format* is used. Valid formats include **decimal**, **octal**, **hex**, **character**, **byte**, **directory**, **i-node**, and **write**. The last creates a file in the current directory (see *FILES*) containing the buffer data.
- callout**
Aliases: **calls**, **call**, **c**, **timeout**, **time**, **tout**.
Print all entries in the callout table.
- map** [list of map names]
Format the named system map structures.
- nm** [list of symbols]
Print symbol value and type as found in the *namelist* file.
- ts** [list of text/data addresses]
Find the closest text or data symbols to the given addresses.
- ds** [list of text/data addresses]
Same function as **ts**.
- od** [symbol name or address] [count] [format]
Aliases: **dump**, **rd**.
Dump *count* data values starting at the symbol value or address given according to *format*. Allowable formats are **octal**, **longoct**, **decimal**, **longdec**, **character**, **hex**, or **byte**.
- odl** [process number in decimal] [address in hex] [count in decimal] [format]
Dump *count* data values for the local space defined by the process number and the address according to *format*. Allowable formats are **octal**, **longoct**, **decimal**, **longdec**, **character**, **hex**, or **byte**. (UNIX System V/286 only.)
- !** Escape to shell.
- q** Exit from *crash*.

CRASH(1M)

? Print synopsis of commands.

ALIASES

There are built-in aliases for many of the *formats* as well as those listed for the commands. Some of them are:

byte	b.
character	char, c.
decimal	dec, e.
directory	direct, dir, d.
hexadecimal	hexadec, hex, h, x.
i-node	ino, i.
longdec	ld, D.
longoct	lo, O.
octal	oct, o.
write	w.

FILES

/usr/include/sys/*.h	header files for table and structure info
/dev/mem	default system image file
/unix	default namelist file
buf.#	files created containing buffer data

SEE ALSO

mount(1M), nm(1), ps(1), sh(1), stty(1).

BUGS

Most flags are abbreviated and will have little meaning to the uninitiated user. A source listing of the system header files at hand would be most useful while using *crash*.

Stack tracing of the current process on a running system does not work.

NAME

cron - clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in crontab files; users can submit their own crontab file via the *crontab* command. Commands which are to be executed only once may be submitted via the *at* command. Since *cron* never exits, it should only be executed once. This is best done by running *cron* from the initialization process through the file */etc/rc* [see *init(8)*].

Cron only examines crontab files and at command files during process initialization and when a file changes. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

FILES

/usr/lib/cron main cron directory
/usr/lib/cron/log accounting information
/usr/spool/cron spool area

SEE ALSO

at(1), *crontab(1)*, *sh(1)*.

DIAGNOSTICS

A history of all actions taken by *cron* are recorded in */usr/lib/cron/log*.

CRONTAB(1)

NAME

crontab — user crontab file

SYNOPSIS

```
crontab [file]
crontab -r
crontab -l
```

DESCRIPTION

Crontab copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs. The *-r* option removes a user's crontab from the crontab directory. *Crontab -l* will list the crontab file for the invoking user.

Users are permitted to use *crontab* if their names appear in the file */usr/lib/cron/cron.allow*. If that file does not exist, the file */usr/lib/cron/cron.deny* is checked to determine if the user should be denied access to *crontab*. If neither file exists, only root is allowed to submit a job. If either file is *at.deny*, global usage is permitted. The allow/deny files consist of one user name per line.

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

```
minute (0-59),
hour (0-23),
day of the month (1-31),
month of the year (1-12),
day of the week (0-6 with 0=Sunday).
```

Each of these patterns may be either an asterisk (meaning all legal values), or a list of elements separated by commas. An element is either a number, or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 * 1 would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, 0 0 * * 1 would run a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by \) is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your \$HOME directory with an *arg0* of *sh*. Users who desire to have their *profile* executed must explicitly do so in the crontab file. *Cron* supplies a default environment for every shell, defining HOME, LOGNAME, SHELL(=/bin/sh), and PATH(=:bin:/usr/bin:/usr/sbin).

NOTE: Users should remember to redirect the standard output and standard error of their commands! If this is not done, any generated output or errors will be mailed to the user.

FILES

/usr/lib/cron	main cron directory
/usr/spool/cron/crontabs	spool area
/usr/lib/cron/log	accounting information
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users

SEE ALSO

sh(1),
cron(1M).

NAME

`csh` - a shell (command interpreter) with C-like syntax

SYNOPSIS

`csh [-cefinstvVxX] [arg ...]`

DESCRIPTION

Csh is a first implementation of a command language interpreter incorporating a history mechanism (see History Substitutions) job control facilities (see Jobs) and a C-like syntax.

Note that although job control facilities are a standard function of Berkeley Csh, Microport System V/AT does not support Berkeley job control at this time.

An instance of *csh* begins by executing commands from the file `'.cshrc'` in the *home* directory of the invoker. If this is a login shell then it also executes commands from the file `'.login'`.

In the normal case, the shell will then begin reading commands from the terminal, prompting with `'%'`. Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and then parsed. Finally each command in the current line is executed.

When a login shell terminates it executes commands from the file `'.logout'` in the users home directory.

Lexical structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters `'&'`, `'|'`, `'<'`, `'>'`, `'('`, `')'` form separate words. If doubled in `'&&'`, `'||'`, `'<<'` or `'>>'` these pairs form single words. These parser metacharacters may be made part of other words, or prevented from their special meaning by preceding them with `'\'`. A new line preceded by a `'\'` is equivalent to a blank.

In addition, strings enclosed in matched pairs of quotations, `'`, `"` or `'''`, form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of `'` or `'''` characters a newline preceded by a `'\'` gives a true newline character.

When the shell's input is not a terminal, the character `'#'` introduces a comment which continues to the end of the input line. It is prevented from this special meaning when preceded by `'\'` and in quotations using `'`, `"`, and `'''`.

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by `'|'` characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by `'&'`, and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an `'&'`.

Any of the above may be placed in `'('` `)'` to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with `'|'` or `'&&'` indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See *Expressions*.)

Jobs

The shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the *jobs* command, and assigns them small integer numbers. When a job is started asynchronously with '&', the shell prints a line which looks like:

```
[1] 1234
```

indicating that the job started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit the key ^Z (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been 'Stopped', and print another prompt. You can then manipulate the state of this job, putting it in the background with the *bg* command, or run some other commands and then eventually bring the job back into the foreground with the foreground command *fg*. A ^Z takes effect immediately, and is like an interrupt in that both pending output and unread input are discarded when it is typed. There is another special key, ^Y, which does not generate a STOP signal until a program attempts to *read(2)* it. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command "stty tostop". If you set this tty option, background jobs will stop when they try to produce output, just as they do when trying to read input.

There are several ways to refer to jobs in the shell. The character '%' introduces a job name. If you wish to refer to job number 1, you can name it '%1'. Naming a job brings it to the foreground; thus '%1' is a synonym for 'fg %1', bringing job 1 back into the foreground. Similarly saying '%1 &' resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them, if these prefixes are unambiguous. Thus, '%ex' would normally restart a suspended *ex(1)* job, if there were only one suspended job whose name began with the string 'ex'. You can also say '%?string', which specifies a job whose text contains *string*, if there is only one such job.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a '+' and the previous job with a '-'. The abbreviation '%+' refers to the current job and '%-' refers to the previous job. For close analogy with the syntax of the *history* mechanism (described below), '%%' is also a synonym for the current job.

Status reporting

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt, so that it does not otherwise disturb your work. If, however, you set the shell variable *notify*, the shell will notify you immediately of changes of status in background jobs. There is also a shell command *notify* which marks a single process so that its status changes will be immediately reported. By default *notify* marks the current process; simply say 'notify' after starting a background job to mark it.

When you try to leave the shell while jobs are stopped, you will be warned that 'You have stopped jobs.' You may use the *jobs* command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the suspended jobs will be terminated.

NOTE: Although job control facilities are a standard function of Berkeley Csh, Microport System V/AT does not support Berkeley job control at this time.

Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur.

History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence. History substitutions begin with the character '!' and may begin anywhere in the input stream (with the provision that they do not nest.) This '!' may be preceded by an '\ ' to prevent its special meaning; for convenience, a '!' is passed unchanged when it is followed by a blank, tab, newline, '=' or '('. (History substitutions also occur when an input line begins with '^'. This special abbreviation will be described later.) Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. The size is controlled by the *history* variable; the previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

For definiteness, consider the following output from the *history* command:

```

9 write michael
10 ex write.c
11 cat oldwrite.c
12 diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing an '!' in the prompt string.

With the current event 13, we can refer to previous events by event number '!11', relatively as '!-2' (referring to the same event), by a prefix of a command word as in '!d' for event 12 or '!wri' for event 9, or by a string contained in a word in the command as in '!?mic?' also referring to event 9. These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case '!!' refers to the previous command; thus '!!' alone is essentially *redo*.

To select words from an event we can follow the event specification by a ':' and a designator for the desired words. The words of an input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. The basic word designators are:

```

0      first (command)word
n      n'th argument
^      first argument, i.e. '1'
$      last argument
%      word matched by (immediately preceding) ?s? search
x-y    range of words
->     abbreviates '0-y'
*      abbreviates '^-$', or nothing if only 1 word in event
x*     abbreviates 'x-$'
x-     like 'x*' but omitting word '$'
```

The ':' separating the event specification from the word designator can be omitted if the argument selector begins with a '^', '\$', '*' '-' or '%'. A sequence of modifiers, each preceded by a ':', can be placed after the optional word designator. The following modifiers are defined:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing '.xxx' component, leaving the root name.
e	Remove all but the extension '.xxx' part.
s/l/r/	Substitute <i>l</i> for <i>r</i>
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g. 'g&'. Print the new command but do not execute it.
p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like q, but break into words at blanks, tabs and newlines.

Unless preceded by a 'g' the modification is applied only to the first modifiable word. With substitutions, it is an error for no word to be applicable.

The left hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of '/'; a '\ quotes the delimiter into the *l* and *r* strings. The character '&' in the right hand side is replaced by the text from the left. A '\ quotes '&' also. A null *l* uses the previous string either from an *l* or from a contextual scan string *s* in '!?s?'. The trailing delimiter in the substitution may be omitted if a newline follows immediately, as may the trailing '?' in a contextual scan.

A history reference may be given without an even specification, e.g. '!\$'. In this case the reference is to the previous command unless a previous history reference occurred on the same line, in which case this form repeats the previous reference. Thus '!?foo^ \$!' gives the first and last arguments from the command matching '?foo?'.

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a '^'. This is equivalent to '!s^' providing a convenient shorthand for substitutions on the text of the previous line. Thus '^lib^lib' fixes the spelling of 'lib' in the previous command. Finally, a history substitution may be surrounded with '{' and '}' if necessary to insulate it from the characters which follow. Thus, after 'ls -ld ~paul' we might do '{3}a' to do 'ls -ld ~paula', while '!3a' would look for a command starting '3a'.

Quotations with ' and "

The quotation of strings by "" and "" are used to prevent all or some of the remaining substitutions. Strings enclosed in "" are prevented from any further interpretation. Strings enclosed in "" are yet variable and command expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see *Command Substitution* below) does a "" quoted string yield parts of more than one word; "" quoted strings never do.

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct conunands and the first word of each conunand, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available, as though that conunand were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for 'ls' is 'ls -l' the command 'ls /usr' would map to 'ls -l usr', the argument list here being undisturbed. Similarly if the alias for 'lookup' was 'grep !^ /etc/passwd' then 'lookup bill' would map to 'grep bill/etc/passwd'.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. If the first word of the new text is the same as the old, looping is prevented by flagging it to prevent further aliasing. Other loops are detected and cause an error message.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can 'alias print 'pr \!* | lpr'' to make a command which *pr's* its arguments to the line printer.

Variable substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the *set* and *unset* conunands. Of the variables referred to by the shell, a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The setting of this variable results from the *-v* command line option.

Other operations treat variables numerically. The '@' command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by '\$' characters. This expansion can be prevented by preceding the '\$' with a '\' except within ""'s where it always occurs, and within "'s where it never occurs. Strings quoted by "" are interpreted later (see *Command substitution* below) so '\$' substitution does not occur there until later, if at all. A '\$' is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the conunand name and entire argument list are expanded together. It is thus possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in "" or given the 'q' modifier, the results of variable substitution may eventually be command and filename substituted. Within ""', a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variables value separated by blanks. When the 'q' modifier is applied to a substitution, the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

`$name`
`$(name)`

Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.

If *name* is not a shell variable, but is set in the environment, then that value is returned (but : modifiers and the other forms given below are not available in this case).

`$name[selector]`
`$(name[selector])`

May be used to select only some of the words from the value of *name*. The selector is subjected to '\$' substitution and may consist of a single number or two numbers separated by a '-'. The first word of a variable's value is numbered '1'. If the first number of a range is omitted it defaults to '1'. If the last member of a range is omitted it defaults to '\$#name'. The selector '*' selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

`$#name`
`$(#name)`

Gives the number of words in the variable. This is useful for later use in a '[selector]'.

`$0`

Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

`$number`
`$(number)`

Equivalent to '\$argv[number]'.

`$*`

Equivalent to '\$argv[*]'.

The modifiers ':h', ':t', ':r', ':q' and ':x' may be applied to the substitutions above as may ':gh', ':gt' and ':gr'. If braces '{ '}' appear in the command form then the modifiers must appear within the braces. The current implementation allows only one ':' modifier on each '\$' expansion.

The following substitutions may not be modified with ':' modifiers.

`$?name`
`$(?name)`

Substitutes the string '1' if name is set, '0' if it is not.

`$?0`

Substitutes '1' if the current input filename is known, '0' if it is not.

`$$`

Substitutes the (decimal) process number of the (parent) shell.

`$<`

Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of built-in commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution

Command substitution is indicated by a command enclosed in ````. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within ````'s, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename substitution

If a word contains any of the characters `*`, `?`, `[` or `{` or begins with the character `~`, then that word is a candidate for filename substitution, also known as 'globbing'. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters `*`, `?` and `[` imply pattern matching; the characters `~` and `{` being more akin to abbreviations.

In matching filenames, the character `.` at the beginning of a filename or immediately following a `/`, as well as the character `/` must be matched explicitly. The character `*` matches any string of characters, including the null string. The character `?` matches any single character. The sequence `[...]` matches any one of the characters enclosed. Within `[...]`, a pair of characters separated by `-` matches any character lexically between the two.

The character `~` at the beginning of a filename is used to refer to home directories. Standing alone, i.e. `~` it expands to the invokers home directory as reflected in the value of the variable `home`. When followed by a name consisting of letters, digits and `-` characters the shell searches for a user with that name and substitutes their home directory; thus `~ken` might expand to `/usr/ken` and `~ken/chmach` to `/usr/ken/chmach`. If the character `~` is followed by a character other than a letter or `/` does not appear at the beginning of a word, it is left undisturbed.

The metanotation `a{b,c,d}e` is a shorthand for `abc ace ade`. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus, `~source/sl/(oldls,ls).c` expands to `/usr/source/sl/oldls.c /usr/source/sl/ls.c` whether or not these files exist, without any chance of error if the home directory for 'source' is `/usr/source`. Similarly `./{memo,*box}` might expand to `./memo ./box ./mbox`. (Note that 'memo' was not sorted with the results of matching `*box`.) As a special case `{, }` and `{ }` are passed undisturbed.

Input/output

The standard input and standard output of a command may be redirected with the following syntax:

`<name`

Open file *name* (which is first variable, command and filename expanded) as the standard input.

<<word

Read the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting '\', "'", '" or "" appears in *word*, variable and command substitution is performed on the intervening lines, allowing '\ to quote '\$', \ and "''. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name

>! name

>& name

>&!name

The file *name* is used as standard output. If the file does not exist then it is created; if the file exists, it is truncated, and previous contents are lost.

If the variable *noclobber* is set, then the file must not exist or be a character special file (e.g. a terminal or '/dev/null') or an error results. This helps prevent accidental destruction of files. In this case the '!' forms can be used and suppress this check.

The forms involving '&' route the diagnostic output into the specified file as well as the standard output. *Name* is expanded in the same way as '<' input filenames are.

>>name

>>& name

>>! name

>>&! name

Uses file *name* as standard output like '>' but places output at the end of the file. If the variable *noclobber* is set, then it is an error for the file not to exist unless one of the '!' forms is given. Otherwise similar to '>'.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The '<<' mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input. Note that the default standard input for a command run detached is not modified to be the empty file '/dev/null'; rather the standard input remains as the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, then the process will block and the user will be notified (see Jobs above).

Diagnostic output may be directed through a pipe with the standard output. Simply use the form '|&' rather than just '|'.

Expressions

A number of the built-in commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the @, *exit*, *if*, and *while* commands. The following operators are available:

|| && | ^ & == != == ! <= >= < > << >> + - * / % ! ~ ()

Here the precedence increases to the right, '=' '!=' '~' and '!~', '<=' '>=' '>' '<<' and '>>', '+', '-' and '.', '*', '/' and '%' being, in groups, at the same level. The '=', '!=', '~' and '!~' operators compare their arguments as strings; all others operate on numbers. The operators '~' and '!~' are like '!=' and '=' except that the right hand side is a *pattern* (containing, e.g. '*'s, '?'s and instances of '[...]') against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings which begin with '0' are considered octal numbers. Null or missing arguments are considered '0'. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser ('&' '|' '<' '>' '(' ')') they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in '(' and ')' and file enquiries of the form '-l name' where *l* is one of:

```

r      read access
w      write access
x      execute access
e      existence
o      ownership
z      zero size
f      plain file
d      directory

```

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries return false, i.e. '0'. Command executions succeed, returning true, i.e. '1', if the command exits with status 0; otherwise they fail, returning false, i.e. '0'. If more detailed status information is required, then the command should be executed outside of an expression and the variable *status* examined.

Control flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The *foreach*, *switch*, and *while* statements, as well as the *if-then-else* form of the *if* statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's will succeed on non-seekable inputs.)

Built-in commands

Built-in commands are executed within the shell. If a built-in command occurs as any component of a pipeline except the last then it is executed in a subshell.

alias

alias name

alias name wordlist

The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name*; *wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias*.

alloc

Shows the amount of dynamic core in use, broken down into used and free core, and address of the last location in the heap. With an argument shows each used and free block on the internal dynamic memory chain indicating its address, size, and whether it is used or free. This is a debugging command and may not work in production versions of the shell; it requires a modified version of the system memory allocator.

bg

bg %job ...

Puts the current or specified jobs into the background, continuing them if they were stopped.

break

Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while*. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw

Causes a break from a *switch*, resuming after the *endsw*.

case label:

A label in a *switch* statement as discussed below.

cd

cd name

chdir

chdir name

Change the shells working directory to a directory *name*. If no argument is given then change to the home directory of the user.

If *name* is not found as a subdirectory of the current directory (and does not begin with '/', './' or './..'), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails, but *name* is a shell variable whose value begins with '/', then this is tried to see if it is a directory.

continue

Continue execution of the nearest enclosing *while* or *foreach*. The rest of the commands on the current line are executed.

default:

Labels the default case in a *switch* statement. The default should come after all *case* labels.

dirs

Prints the directory stack; the top of the stack is at the left, the first directory in the stack being the current directory.

echo wordlist

echo -n wordlist

The specified words are written to the shells standard output, separated by spaces, and terminated with a newline unless the **-n** option is specified.

else

end

endif

endsw

See the description of the *foreach*, *if*, *switch*, and *while* statements below.

eval arg ...

(As in *sh(1)*.) The arguments are read as input to the shell and the resulting conunand(s) executed. This is usually used to execute commands generated as the result of conunand or variable substitution, since parsing occurs before these substitutions. See *tset(1)* for an example of using *eval*.

exec command

The specified conunand is executed in place of the current shell.

exit

exit(*expr*)

The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

fg

fg %job ...

Brings the current or specified jobs into the foreground, continuing them if they were stopped.

foreach name (wordlist)

...
end

The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching *end* are executed. (Both *foreach* and *end* must appear alone on separate lines.)

The built-in command *continue* may be used to continue the loop prematurely and the built-in command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with '?' before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

glob wordlist

Like *echo* but no '\` escapes are recognized and words are delimited by null characters in the output. Useful for programs that wish to use the shell to filename expand a list of words.

gotoword

The specified *word* is filename and command expanded to yield a string of the form 'label'. The shell rewinds its input as much as possible and searches for a line of the form 'label:' possibly preceded by blanks or tabs. Execution continues after the specified line.

hashstat

Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding *exec's*). An *exec* is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component which does not begin with a '/

history

history *n*

history -r *n*

Displays the history event list; if *n* is given only the *n* most recent events are printed. The -r option reverses the order of printout to be most recent first rather than oldest first.

if (*expr*) command

If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple conunand, not a pipeline, a conunand list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is not executed (this is a bug).

```

if (expr) then
...
else if (expr2) then
...
else
...
endif

```

If the specified *expr* is true then the commands to the first *else* are executed; else if *expr2* is true then the commands to the second *else* are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The words *else* and *endif* must appear at the beginning of input lines; the *if* must appear alone on its input line or after *anelse*.)

```

jobs
jobs -l

```

Lists the active jobs; given the *-l* options lists process id's in addition to the normal information.

```

kill %job
kill -sig %job ...
kill pid
kill -sig pid ...
kill -l

```

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in *usr/include/signal.h*, stripped of the prefix "SIG"). The signal names are listed by "kill -l". There is no default, saying just 'kill' does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal as well.

limit

limit resource

limit resource maximum-use

Limits the consumption by the current process and each process it creates to not individually exceed *maximum-use* on the specified *resource*. If no *maximum-use* is given, then the current limit is printed; if no *resource* is given, then all limitations are given.

Resources controllable currently include *cpulime* (the maximum number of cpu-seconds to be used by each process), *filesize* (the largest single file which can be created), *datasize* (the maximum growth of the data+stack region via *strk(2)* beyond the end of the program text), *stacksize* (the maximum size of the automatically-extended stack region), and *coredumpsize* (the size of the largest core dump that will be created).

The *maximum-use* may be given as a (floating point or integer) number followed by a scale factor. For all limits other than *cpulime* the default scale is 'k' or 'kilobytes' (1024 bytes); a scale factor of 'm' or 'megabytes' may also be used. for *cpulime* the default scaling is 'seconds', while 'm' for minutes or 'h' for hours, or a time of the form 'mm:ss' giving minutes and seconds may be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

login

Terminate a login shell, replacing it with an instance of */bin/login*. This is one way to log off, included for compatibility with *sh(1)*.

logout

Terminate a login shell. Especially useful if *ignoreeof* is set.

newgrp

Changes the group identification of the caller; for details see *newgrp(1)*. A new shell is executed by *newgrp* so that the shell state is lost.

nice

nice +number

nice command

nice +number command

The first form sets the *nice* for this shell to 4. The second form sets the *nice* to the given number. The final two forms run *command* at priority 4 and *number* respectively. The super-user may specify negative niceness by using '*nice -number ...*'. *Command* is always executed in a sub-shell, and the restrictions placed on commands in simple *if* statements apply.

nohup

nohup command

The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with '&' are effectively *nohup*'ed.

notify

notify %job ...

Causes the shell to notify the user asynchronously when the status of the current or specified jobs changes; normally notification is presented before a prompt. This is automatic if the shell variable *notify* is set.

onintr

onintr -

onintr label

Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts, which is to terminate shell scripts or to return to the terminal command input level. The second form '*onintr -*' causes all interrupts to be ignored. The final form causes the shell to execute a '*goto label*' when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running detached and interrupts are being ignored, all forms of *onintr* have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

popd

popd +n

Pops the directory stack, returning to the new top directory. With an argument '*+n*' discards the *n*th entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

pushd

pushd name

pushd +n

With no arguments, *pushd* exchanges the top two elements of the directory stack. Given a *name* argument, *pushd* changes to the new directory (ala *cd*) and pushes the old current working directory (as in *csw*) onto the directory stack. With a numeric argument, rotates the *n*th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

rehash

Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command

The specified *command* which is subject to the same restrictions as the *command* in the one line *if* statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

set**set name**

set name=word

set name{index}=word

set name=(wordlist)

The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *index*'th component of name to *word*; this component must already exist. The final form sets *name* to the list of words in *wordlist*. In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note, however, that variable expansion happens for all arguments before any setting occurs.

setenv name value

Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variable USER, TERM, and PATH are automatically imported to and exported from the *cs*h variables *user*, *term*, and *path*; there is no need to use *setenv* for these.

shift**shift variable**

The members of *argv* are shifted to the left, discarding *argv* [1]. It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

source name

The shell reads commands from *name*. *Source* commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands. Input during *source* commands is never placed on the history list.

stop

stop %job ...

Stops the current or specified job executing in the background

suspend

Causes the shell to stop in its tracks, much as if it had been sent a stop signal with ^Z. This is most often used to stop shells started by *su*(1).

switch (string)

case str1:

...

breaksw

...

default:

...

breaksw

endsw

Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters '*', '?' and '['...' may be used in the case labels, which are variable expanded. If none of the labels match before a 'default' label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

time

time command

With no argument, a summary of time used by this shell and its children is printed. If arguments are given, the specified simple command is timed and a time summary as described under the *time* variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

umask

umask value

The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others, or 022 giving read and execute access for users in the group or others.

unalias pattern

All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by 'unalias*'. It is not an error for nothing to be *unaliased*.

unhash

Use of the internal hash table to speed location of executed programs is disabled.

unlimit resource

unlimit

Removes the limitation on *resource*. If no *resource* is specified, then all *resource* limitations are removed.

unset pattern

All variables whose names match the specified pattern are removed. Thus all variables are removed by 'unset*'; this has noticeably distasteful side-effects. It is not an error for nothing to be *unset*.

unsetenv pattern

Removes all variables whose name match the specified pattern from the environment. See also the *setenv* command above and *printenv*(1).

wait

All background jobs are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (expr)

...
end

While the specified expression evaluates non-zero, the commands between the *while* and the matching *end* are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. (The *while* and *end* must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

%job

Brings the specified job into the foreground.

%job &

Continues the specified job in the background.

@

@ name = expr

@ name[index] = expr

The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr*. If the expression contains '<', '>', '&' or '|' then at least this part of the expression must be placed within '(' ')'. The third form assigns the value of *expr* to the *index*'th argument of *name*. Both *name* and its *index*'th component must already exist.

The operators '*=', '+=', etc are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* which would otherwise be single words.

Special postfix '++' and '--' operators increment and decrement *name* respectively, i.e. '@ i++'.

Pre-defined and environment variables

The following variables have special meaning to the shell. Of these, *argv*, *cwd*, *home*, *path*, *prompt*, *shell* and *status* are always set by the shell. Except for *cwd* and *status* this setting occurs only at initialization; these variables will not then be modified unless this is done explicitly by the user.

This shell copies the environment variable *USER* into the variable *user*, *TERM* into *term*, and *HOME* into *home*, and copies these back into the environment whenever the normal shell variables are reset. The environment variable *PATH* is likewise handled; it is not necessary to worry about its setting other than in the file *.cshrc*, as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it. (It could be set once in the *.login* except that commands through *net(1)* would not see the definition.)

argv	Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e. '\$1' is replaced by '\$argv[1]', etc.
cdpath	Gives a list of alternate directories searched to find subdirectories in <i>chdir</i> commands.
cwd	The full pathname of the current directory.
echo	Set when the <i>-x</i> command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-built-in commands all expansions occur before echoing. Built-in commands are echoed before command and filename substitution, since these substitutions are then done selectively.

history	Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. Too large values of <i>history</i> may run the shell out of memory. The last executed command is always saved on the history list.
home	The home directory of the invoker, initialized from the environment. The filename expansion of '~' refers to this variable.
ignoreeof	If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-D's.
mail	The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says 'You have new mail.' if the file exists with an access time not greater than its modify time. If the first word of the value of <i>mail</i> is numeric it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes. If multiple mail files are specified, then the shell says 'New mail in <i>name</i> ' when there is mail in the file <i>name</i> .
noclobber	As described in the section on <i>Input/output</i> , restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that '>>' redirections refer to existing files.
noglob	If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.
nonomatch	If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e. 'echo [' still gives an error.
notify	If set, the shell notifies asynchronously of job completions. The default is to rather present job completions just before printing a prompt.
path	Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no <i>path</i> variable then only full path names will execute. The usual search path is '.', '/bin' and '/usr/bin', but this may vary from system to system. For the super-user the default search path is '/etc', '/bin' and '/usr/bin'. A shell which is given neither the <code>-c</code> nor the <code>-t</code> option will normally hash the contents of the directories in the <i>path</i> variable after reading <i>.cshrc</i> , and each time the <i>path</i> variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the <i>rehash</i> or the commands may not be found.
prompt	The string which is printed before each command is read from an interactive terminal input. If a '!' appears in the string it will be replaced by the current event number unless a preceding '\ ' is given. Default is '%', or '#' for the super-user.
shell	The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of <i>Non-built-in command Execution</i> below.) Initialized to the (system-dependent) home of the shell.

- status** The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Built-in commands which fail return exit status '1', all other built-in commands set status '0'.
- time** Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.
- verbose** Set by the `-v` command line option, causes the words of each command to be printed after history substitution.

Non-built-in command execution

When a command to be executed is found to not be a built-in command the shell attempts to execute the command via `exec(2)`. Each word in the variable `path` names a directory from which the shell will attempt to execute the command. If it is given neither a `-c` nor a `-f` option, the shell will hash the names in these directories into an internal table so that it will only try an `exec` in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via `unhash`), or if the shell was given a `-c` or `-f` argument, and in any case for each directory component of `path` which does not begin with a '/', the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus `(cd ; pwd) ; pwd` prints the *home* directory; leaving you where you were (printing this after the home directory), while `cd ; pwd` leaves you in the *home* directory. Parenthesized commands are most often used to prevent `chdir` from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an *alias* for *shell* then the words of the alias will be prepended to the argument list to form the shell command. The first word of the *alias* should be the full path name of the shell (e.g. '\$shell'). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

Argument list processing

If argument 0 to the shell is '-' then this is a login shell. The flag arguments are interpreted as follows:

- c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in `argv`.
- e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.
- f The shell will start faster, because it will neither search for nor execute commands from the file '.cshrc' in the invokers home directory.
- l The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.
- n Commands are parsed, but not executed. This may aid in syntactic checking of shell scripts.

CSH(1)

- s Command input is taken from the standard input.
- t A single line of input is read and executed. A '\ ' may be used to escape the newline at the end of this line and continue onto another line.
- v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.
- x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.
- V Causes the *verbose* variable to be set even before 'cshrc' is executed.
- X Is to -x as -V is to -v.

After processing of flag arguments if arguments remain but none of the -c, -l, -s, or -t options was given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by '\$0'. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute such a 'standard' shell if the first character of a script is a ':' or a newline. Remaining arguments initialize the variable *argv*.

Signal handling

The shell normally ignores *quit* signals. Jobs running detached (either by '&' or the *bg* or *%... &* commands) are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file '.logout'.

NOTE: Although job control facilities are a standard function of Berkeley Csh, Microport System V/AT does not support Berkeley job control at this time.

AUTHOR

William Joy. Job control and directory stack features first implemented by J.E. Kulp of I.I.A.S.A, Laxenburg, Austria, with different syntax than that used now.

FILES

~/cshrc	Read at beginning of execution by each shell.
~/login	Read by login shell, after 'cshrc' at login.
~/logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a '#'.
/tmp/sh*	Temporary file for '<<'.
/etc/passwd	Source of home directories for '~name'.

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves filename expansion is limited to 1/6th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

SEE ALSO

sh(1), access(2), exec(2), fork(2), pipe(2), umask(2), wait(2), tty(4), a.out(5), environ(5), 'An introduction to the C shell'

BUGS

When a command is restarted from a stop, the shell prints the directory it started in if this is different from the current directory; this can be misleading (i.e. wrong) as the job may have changed directories internally.

Shell built-in functions are not stoppable/restartable. Command sequences of the form 'a ; b ; c' are also not handled gracefully when stopping is attempted. If you suspend 'b', the shell will then immediately execute 'c'. This is especially noticeable if this expansion results from an *alias*. It suffices to place the sequence of commands in ()'s to force it to a subshell, i.e. '(a ; b ; c)'.

Control over tty output after processes are started is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by '?', are not placed in the *history* list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with '|', and to be used with '&' and ';' metasyntax.

It should be possible to use the ':' modifiers on the output of command substitutions. All and more than one ':' modifier should be allowed on '\$' substitutions.

CSPLIT(1)

NAME

`csplit` - context split

SYNOPSIS

`csplit [-s] [-k] [-f prefix] file arg1 [.. argn]`

DESCRIPTION

Csplit reads *file* and separates it into *n*+1 sections, defined by the arguments *arg1*... *argn*. By default the sections are placed in `xx00` ... `xxn` (*n* may not be greater than 99). These sections get the following pieces of *file*:

- 00: From the start of *file* up to (but not including) the line referenced by *arg1*.
- 01: From the line referenced by *arg1* up to the line referenced by *arg2*.
- .
- .
- .
- n*+1: From the line referenced by *argn* to the end of *file*.

If the *file* argument is a `-` then standard input is used.

The options to *csplit* are:

- `-s` *Csplit* normally prints the character counts for each file created. If the `-s` option is present, *csplit* suppresses the printing of all character counts.
- `-k` *Csplit* normally removes created files if an error occurs. If the `-k` option is present, *csplit* leaves previously created files intact.
- `-f prefix` If the `-f` option is used, the created files are named *prefix00* ... *prefixn*. The default is `xx00` ... `xxn`.

The arguments (*arg1* ... *argn*) to *csplit* can be a combination of the following:

- /rexp/* A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional `+` or `-` some number of lines (e.g., */Page/-5*).
- %rexp%* This argument is the same as */rexp/*, except that no file is created for the section.
- lnno* A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.
- {num}* Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the Shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. *Csplit* does not affect the original file; it is the users responsibility to remove it.

EXAMPLES

```
csplit -f cobol file '/procedure division/' /par5./ /par16./
```

This example creates four files, `cobol00` ... `cobol03`. After editing the "split" files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

```
csplit -k file 100 {99}
```

This example would split the file at every 100 lines, up to 10,000 lines. The `-k` option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c '%main(%' '/'^)/+1' {20}
```

Assuming that `prog.c` follows the normal C coding convention of ending routines with a `}` at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in `prog.c`.

SEE ALSO

`ed(1)`, `sh(1)`,
`regex(5)` in the Software Development System manual.

DIAGNOSTICS

Self explanatory except for:

`arg` - out of range

which means that the given argument did not reference a line between the current position and the end of the file.

CT(1C)

NAME

`ct` - spawn `getty` to a remote terminal

SYNOPSIS

`ct` [`-h`] [`-v`] [`-wn`] [`-speed`] `telno` ...

DESCRIPTION

`Ct` dials the phone number of a modem that is attached to a terminal, and spawns a `getty` process to that terminal. `Telno` is a telephone number, with equal signs for secondary dial tones and minus signs for delays at appropriate places. If more than one telephone number is specified, `ct` will try each in succession until one answers; this is useful for specifying alternate dialing paths.

`Ct` will try each line listed in the file `/usr/lib/uucp/L-devices` until it finds an available line with appropriate attributes or runs out of entries. If there are no free lines, `ct` will ask if it should wait for one, and if so, for how many minutes it should wait before it gives up. `Ct` will continue to try to open the dialers at one-minute intervals until the specified limit is exceeded. The dialogue may be overridden by specifying the `-wn` option, where `n` is the maximum number of minutes that `ct` is to wait for a line.

Normally, `ct` will hang up the current line, so that that line can answer the incoming call. The `-h` option will prevent this action. If the `-v` option is used, `ct` will send a running narrative to the standard error output stream.

The data rate may be set with the `-s` option, where `speed` is expressed in baud. The default rate is 300.

After the user on the destination terminal logs out, `ct` prompts, `Reconnect?` If the response begins with the letter `n` the line will be dropped; otherwise, `getty` will be started again and the `login:` prompt will be printed.

Of course, the destination terminal must be attached to a modem that can answer the telephone.

FILES

`/usr/lib/uucp/L-devices`
`/usr/adm/ctlog`

SEE ALSO

`cu(1C)`, `login(1)`, `uucp(1C)`.

NAME

ctrace — C program debugger

SYNOPSIS

ctrace [options] [file]

DESCRIPTION

Ctrace allows you to follow the execution of a C program, statement by statement. The effect is similar to executing a shell procedure with the `-x` option. *Ctrace* reads the C program in *file* (or from standard input if you do not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of *ctrace* into a temporary file because the `cc(1)` command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the `bfs(1)` or `tail(1)` commands.

The only *options* you will commonly use are:

- `-f functions` Trace only these *functions*.
- `-v functions` Trace all but these *functions*.

You may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. You can request that variables be printed in additional formats, if appropriate, with these *options*:

- `-o` Octal
- `-x` Hexadecimal
- `-u` Unsigned
- `-e` Floating point

These *options* are used only in special circumstances:

- `-l n` Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.
- `-s` Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the `=` operator in place of the `==` operator.
- `-t n` Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The Diagnostics section explains when to use this option.
- `-P` Run the C preprocessor on the input before tracing it. You can also use the `-D`, `-I`, and `-U cc(1)` preprocessor options.

These *options* are used to tailor the run-time trace package when the traced program will run in a non-UNIX system environment:

- `-b` Use only basic functions in the trace code, that is, those in `ctype(3C)`, `printf(3S)`, and `string(3C)`. These are usually available even in cross-compilers for microprocessors. In particular, this option is needed when

CTRACE(1)

the traced program runs under an operating system that does not have *signal(2)*, or *setjmp(3C)*.

- p 's' Change the trace print function from the default of 'printf('. For example, 'fprintf(stderr,' would send the trace to the standard error output.
- r f Use file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the -p option).

EXAMPLE

If the file *lc.c* contains this C program:

```
1 #include <stdio.h>
2 main()      /* count lines in input */
3 {
4     int c, nl;
5
6     nl = 0;
7     while ((c = getchar()) != EOF)
8         if (c == '\n')
9             ++nl; 10     printf("%d\n", nl); 11 }
```

and you enter these commands and test data: `cc lc.c a.out I (cntl-d)`, the program will be compiled and executed. The output of the program will be the number 2, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke *ctrace* with these commands: `ctrace lc.c >temp.c cc temp.c a.out` the output will be:

```
2 main()
6     nl = 0;
  /* nl == 0 */
7     while ((c = getchar()) != EOF) The program is now waiting for input.
```

If you enter the same test data as before, the output will be:

```
  /* c == 49 or '1' */
8         if (c == '\n')
  /* c == 10 or '\n' */
9             ++nl;
  /* nl == 1 */
7     while ((c = getchar()) != EOF)
  /* c == 10 or '\n' */
8         if (c == '\n')
  /* c == 10 or '\n' */
9             ++nl;
  /* nl == 2 */
7     while ((c = getchar()) != EOF) If you now enter an end of file character (cntl-d) the final output will be:
  /* c == -1 */ 10     printf("%d\n", nl);
  /* nl == 2 */2     return
```

Note that the program output printed at the end of the trace line for the *nl* variable. Also note the `return` comment added by *ctrace* at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable *c* is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this `if` statement, you will probably realize that you used the assignment operator (=) in place of the equal operator (==). You can easily miss this error during code reading.

EXECUTION-TIME TRACE CONTROL

The default operation for *ctrace* is to trace the entire program file, unless you use the -f or -v options to trace specific functions. This does not give you

statement-by-statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program.

You can do both of these by adding `ctroff()` and `ctron()` function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with `if` statements, and you can even conditionally include this code because `ctrace` defines the CTRACE preprocessor variable. For example:

```
#ifdef CTRACE
    if (c == '?' && i > 1000)
        ctron();
#endif
```

You can also call these functions from `sdb(1)` if you compile with the `-g` option. For example, to trace all but lines 7 to 10 in the main function, enter:

```
sdb a.out
main:7b ctroff()
main:11b ctron()
r
```

You can also turn the trace off and on by setting static variable `tr_ct` to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly.

DIAGNOSTICS

This section contains diagnostic messages from both `ctrace` and `cc(1)`, since the traced code often gets some `cc` warning messages. You can get `cc` error messages in some rare cases, all of which can be avoided.

Ctrace Diagnostics

warning: some variables are not traced in this statement

Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the `-t` option to increase this number.

warning: statement too long to trace

This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

cannot handle preprocessor code, use -P option

This is usually caused by `#ifdef/#endif` preprocessor statements in the middle of a C statement, or by a semicolon at the end of a `#define` preprocessor statement.

'if ... else if sequence too long

Split the sequence by removing an `else` from the middle.

possible syntax error, try -P option

Use the `-P` option to preprocess the `ctrace` input, along with any appropriate `-D`, `-I`, and `-U` preprocessor options. If you still get the error message, check the Warnings section below.

Cc Diagnostics

warning: floating point not implemented

warning: illegal combination of pointer and integer

warning: statement not reached

warning: sizeof returns 0

Ignore these messages.

CTRACE(1)

compiler takes size of function

See the *ctrace* "possible syntax error" message above.

yacc stack overflow

See the *ctrace* "if ... else if" sequence too long" message above.

out of tree space; simplify expression

Use the *-t* option to reduce the number of traced variables per statement from the default of 10. Ignore the "ctrace: too many variables to trace" warnings you will now get.

redeclaration of signal

Either correct this declaration of *signal*(2), or remove it and *#include <signal.h>*.

unimplemented structure assignment

This is caused by a bug in the C compiler for the PDP-11.

offset xxxx in control section ...

This is caused by a problem in the current UNIX System/370 C compiler. Use the *cc*(1) *-b2,2* option.

expression causes compiler loop: try simplifying

This is caused by a bug in the UNIX System/370 C compiler. Unfortunately, the only way to avoid it is to use the *ctrace -v* option to not trace the function containing this line.

WARNINGS

You will get a *ctrace* syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

Ctrace assumes that *BADMAG* is a preprocessor macro, and that *EOF* and *NULL* are *#defined* constants. Declaring any of these to be variables, e.g. "int *EOF*;", will cause a syntax error.

BUGS

Ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. *Ctrace* may choose to print the address of an aggregate or use the wrong format (e.g., *%e* for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multi file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

FILES

runtime.c run-time trace package

SEE ALSO

signal(2), *ctype*(3C), *printf*(3S), *setjump*(3C) in the Software Development System manual.

NAME

cu — call another UNIX system

SYNOPSIS

```
cu [-sspeed] [-lline] [-b] [-t] [-d] [-m] [-o] [-c] [-n] telno
| systemname | dir
```

DESCRIPTION

Cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of ASCII files.

cu accepts the following options and arguments.

-sspeed

Specifies the transmission speed (110, 150, 300, 600, 1200, 4800, 9600); 300 is the default value. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

-lline

Specifies a device name to use as the communication line. This can be used to override searching for the first available line having the right speed. When the **-l** option is used without the **-s** option, the speed of a line is taken from the file `/usr/lib/uucp/L-devices`. When the **-l** and **-s** options are used simultaneously, cu will search the L-devices file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise an error message will be printed and the call will not be made. The specified device is generally a directly connected asynchronous line (e.g., `/dev/ttyab`); in this case a phone number is not required but the string `dir` may be used to specify a null acu. If the specified device is associated with an auto dialer, a phone number must be provided.

-h

Emulates local echo, supporting calls to other computer systems which expect terminals to be set to half-duplex mode.

-t

Used when dialing an ASCII terminal which has been set to auto answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.

-d

Causes diagnostic traces to be printed.

-c

Designates that even parity is to be generated for data sent to the remote.

-o

Designates that odd parity is to be generated for data sent to the remote.

-m

Designates a direct line which has modem control.

-n

Will request the phone number to be dialed from the user rather than taking it from the command line.

telno

When using an automatic dialer, the argument is the telephone number with equal signs for secondary dial tone or minus signs for delays, at appropriate places.

systemname

A **uucp** system name may be used rather than a phone number; in this case, cu will obtain an appropriate direct line or phone number from `/usr/lib/uucp/L.sys` (the appropriate baud rate is also read along with phone numbers). Cu will try each phone number or direct line for **systemname** in the L.sys file until a connection is made or all the entries are tried.

dir

Using *dir* insures that cu will use the line specified by the **-l** option.

After making the connection, *cu* runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with `~`, passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with `~`, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with `~` have special meanings.

The *transmit* process interprets the following:

<code>~</code>	terminate the conversation.
<code>~!</code>	escape to an interactive shell on the local system.
<code>~!cmd...</code>	run <i>cmd</i> on the local system (via <code>sb -c</code>).
<code>~\$cmd...</code>	run <i>cmd</i> locally and send its output to the remote system.
<code>~% cd</code>	change the directory on the local system. NOTE: <code>~ed</code> will cause the command to be run by a sub-shell; probably not what was intended.
<code>~%take from [to]</code>	copy file <i>from</i> (on the remote system) to file <i>to</i> on the local system. If <i>to</i> is omitted, the <i>from</i> argument is used in both places.
<code>~%put from [to]</code>	copy file <i>from</i> (on local system) to file <i>to</i> on remote system. If <i>to</i> is omitted, the <i>from</i> argument is used in both places.
<code>~...</code>	send the line <code>~...</code> to the remote system.
<code>~%break</code>	transmit a BREAK to the remote system.
<code>~%nostop</code>	toggles between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output. A line from the remote that begins with `~>` initiates an output diversion to a file. The complete sequence is:

```
~>[>];file
zero or more lines to be written to file
~>
```

Data from the remote is diverted (or appended, if `>>` is used) to *file*. The trailing `~>` terminates the diversion.

The use of `~%put` requires *stty(1)* and *cat(1)* on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of `~%take` requires the existence of *echo(1)* and *cat(1)* on the remote system. Also, `stty tabs` mode should be set on the remote system if tabs are to be copied without expansion.

When `cu` is used on system `X` to connect to system `Y` and subsequently used on system `Y` to connect to system `Z`, commands on system `Y` can be executed by using `~`. For example, `uname` can be executed on `Z`, `X`, and `Y` as follows:

```
uname
Z
~!uname
X
~~!uname
Y
```

In general, `~` causes the command to be executed on the original machine, `~~` causes the command to be executed on the next machine in the chain.

EXAMPLES

To dial a system whose number is 9 201 555 1212 using 1200 baud:
`cu -s1200 9=2015551212`

If the speed is not specified, 300 is the default value.

To login to a system connected by a direct line:
`cu -l /dev/ttyXX dir`

To dial a system with the specific line and a specific speed:
`cu -s1200 -l /dev/ttyXX dir`

To dial a system using a specific line:
`cu -l /dev/culXX 2015551212`

To use a system name:
`cu YYYZZZ`

FILES

```
/usr/lib/uucp/L.sys
/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..(tty-device)
/dev/null
```

SEE ALSO

`cat(1)`, `ct(1C)`, `echo(1)`, `stty(1)`, `uname(1)`, `uucp(1C)`.

DIAGNOSTICS

Exit code is zero for normal exit, non-zero (various values) otherwise.

BUGS

`Cu` buffers input internally.

There is an artificial slowing of transmission by `cu` during the `~%put` operation so that loss of data is unlikely.

CUT(1)

NAME

cut - cut out selected fields of each line of a file

SYNOPSIS

cut -c *list* [*file1 file2 ...*]

cut -f *list* [-*d* *char*] [-*s*] [*file1 file2 ...*]

DESCRIPTION

Use *cut* to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, i.e., character positions as on a punched card (-*c* option) or the length can vary from line to line and be marked with a field delimiter character like *tab* (-*f* option). *Cut* can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

- list* A comma-separated list of integer field numbers (in increasing order), with optional - to indicate ranges as in the -o option of *nroff/troff* for page ranges; e.g., 1,4,7; 1-3,8; -5,10 (short for 1-5,10); or 3- (short for third through last field).
- c* *list* The *list* following -*c* (no space) specifies character positions (e.g., -c1-72 would pass the first 72 characters of each line).
- f* *list* The *list* following -*f* is a list of fields assumed to be separated in the file by a delimiter character (see -*d*); e.g., -f1,7 copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless -*s* is specified.
- d* *char* The character following -*d* is the field delimiter (-*f* option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.
- s* Suppresses lines with no delimiter characters in case of -*f* option. Unless specified, lines with no delimiters will be passed through untouched.

Either the -*c* or -*f* option must be specified.

HINTS

Use *grep*(1) to make horizontal "cuts" (by context) through a file, or *paste*(1) to put files together column-wise (i.e., horizontally). To reorder columns in a table, use *cut* and *paste*.

EXAMPLES

cut -d: -f1,5 /etc/passwd mapping of user IDs to names
name=who am i | cut -f1 -d" " to set *name* to current login name.

DIAGNOSTICS

- line too long* A line can have no more than 1023 characters or fields.
- bad list for c/f option* Missing -*c* or -*f* option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.
- no fields* The *list* is empty.

SEE ALSO

grep(1), *paste*(1).

NAME

`cxref` - generate C program cross-reference

SYNOPSIS

`cxref` [options] files

DESCRIPTION

Cxref analyzes a collection of C files and attempts to build a cross-reference table. *Cxref* utilizes a special version of *cpp* to include `#define`'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the `-c` option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

In addition to the `-D`, `-I` and `-U` options [which are identical to their interpretation by `cc(1)`], the following *options* are interpreted by *cxref*:

`-c` Print a combined cross-reference of all input files.

`-w<num>`

Width option which formats output no wider than `<num>` (decimal) columns. This option will default to 80 if `<num>` is not specified or is less than 51.

`-o file` Direct output to named *file*.

`-s` Operate silently; does not print input file names.

`-t` Format listing for 80-column width.

FILES

`/usr/lib/xcpp` special version of C-preprocessor.

SEE ALSO

`cc(1)`.

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files, anyway.

BUGS

Cxref considers a formal argument in a `#define` macro definition to be a declaration of that symbol. For example, a program that `#includes ctype.h`, will contain many declarations of the variable `c`.

DATE(1)

NAME

date — print and set the date

SYNOPSIS

date [mmddhhmm[yy]] [+format]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24-hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. *date* takes care of the conversion to and from local standard and daylight time.

If the argument begins with +, the output of *date* is under the control of the user. The format for the output is similar to that of the first argument to *printf*(3S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year — 01 to 12
d	day of month — 01 to 31
y	last 2 digits of year — 00 to 99
D	date as mm/dd/yy
H	hour — 00 to 23
M	minute — 00 to 59
S	second — 00 to 59
T	time as HH:MM:SS
j	day of year — 001 to 366
w	day of week — Sunday = 0
a	abbreviated weekday — Sun to Sat
h	abbreviated month — Jan to Dec
r	time in AM/PM notation

EXAMPLE

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
would have generated as output:
DATE: 08/01/76
TIME: 14:45:05
```

DIAGNOSTICS

<i>No permission</i>	if you are not the super-user and you try to change the date;
<i>bad conversion</i>	if the date set is syntactically incorrect;
<i>bad format character</i>	if the field descriptor is not recognizable.

FILES

/dev/kmem

SEE ALSO

printf(3S) in the Software Development System manual.

WARNING

It is a bad practice to change the date while the system is running multiuser.

NAME

dc - desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. [See *bc*(1), a preprocessor for *dc* that provides infix notation and a C-like syntax that implements functions. *Bc* also provides reasonable control structures for programs.] The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number

The value of the number is pushed on the stack. A number is an unbroken string of the digits 0-9. It may be preceded by an underscore () to input a negative number. Numbers may contain decimal points.

+ - / * % ^

The top two values on the stack are added (+), subtracted (-), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

sx The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the *s* is capitalized, *x* is treated as a stack and the value is pushed on it.

lx The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the *l* is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d The top value on the stack is duplicated.

p The top value on the stack is printed. The top value remains unchanged. **P** interprets the top of the stack as an ASCII string, removes it, and prints it.

f All values on the stack are printed.

q exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

x treats the top element of the stack as a character string and executes it as a string of *dc* commands.

X replaces the number on the top of the stack with its scale factor.

[...] puts the bracketed ASCII string onto the top of the stack.

<x >x =x

The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.

v replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

! interprets the rest of the line as a UNIX system command.

c All values on the stack are popped.

DC(1)

- i** The top value on the stack is popped and used as the number radix for further input. **I** pushes the input base on the top of the stack.
- o** The top value on the stack is popped and used as the number radix for further output.
- O** pushes the output base on the top of the stack.
- k** the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
- z** The stack level is pushed onto the stack.
- Z** replaces the number on the top of the stack with its length.
- ?** A line of input is taken from the input source (usually the terminal) and executed.
- ;** **:** are used by *bc* for array operations.

EXAMPLE

This example prints the first ten values of *n!*:

```
[la]+dsa*pla10>y}sy
0sal
lyx
```

SEE ALSO

bc(1).

DIAGNOSTICS

x is unimplemented

where *x* is an octal number.

stack empty

for not enough elements on the stack to do what was asked.

Out of space

when the free list is exhausted (too many digits).

Out of headers

for too many numbers being kept around.

Out of pushdown

for too many items on the stack.

Nesting Depth

for too many levels of nested execution.

NAME

`dcopy` — copy file systems for optimal access time

SYNOPSIS

`/etc/dcopy [-sX] [-an] [-d] [-v] [-ffsize[:isize]] inputfs outputfs`

DESCRIPTION

Dcopy copies file system *inputfs* to *outputfs*. *Inputfs* is the existing file system; *outputfs* is an appropriately sized file system, to hold the reorganized result. For best results *inputfs* should be the raw device and *outputfs* should be the block device. *Dcopy* should be run on unmounted file systems (in the case of the root file system, copy to a new pack). With no arguments, *dcopy* copies files from *inputfs* compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap. The possible options are

- `-sX` supply device information for creating an optimal organization of blocks in a file. The forms of *X* are the same as the `-s` option of *fsck* (1M).
- `-an` place the files not accessed in *n* days after the free blocks of the destination file system (default for *n* is 7). If no *n* is specified then no movement occurs.
- `-d` leave order of directory entries as is (default is to move sub-directories to the beginning of directories).
- `-v` currently reports how many files were processed, and how big the source and destination freelists are.
- `-ffsize[:isize]` specify the *outputfs* file system and inode list sizes (in blocks). If the option (or *isize*) is not given, the values from the *inputfs* are used.

Dcopy catches interrupts and quits and reports on its progress. To terminate *dcopy* send a quit signal, and *dcopy* will no longer catch interrupts or quits.

SEE ALSO

fsck(1M), *mkfs*(1M), *ps*(1).

DD(1)

NAME

dd — convert and copy a file

SYNOPSIS

dd [*option*=*value*] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

<i>option</i>	<i>values</i>
if = <i>file</i>	input file name; standard input is default
of = <i>file</i>	output file name; standard output is default
ibs = <i>n</i>	input block size <i>n</i> bytes (default 512)
obs = <i>n</i>	output block size (default 512)
bs = <i>n</i>	set both input and output block size, superseding <i>ibs</i> and <i>obs</i> ; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs = <i>n</i>	conversion buffer size
skip = <i>n</i>	skip <i>n</i> input blocks before starting copy
seek = <i>n</i>	seek <i>n</i> blocks from beginning of output file before copying
count = <i>n</i>	copy only <i>n</i> input blocks
conv = <i>ascii</i>	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetic to lowercase
ucase	map alphabetic to uppercase
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input block to <i>ibs</i>
... , ...	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if *ascii* or *ebcdic* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and new-line added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output block of size *cbs*.

After completion, *dd* reports the number of whole and partial input and output blocks.

EXAMPLE

This command will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the ASCII file *x*:

```
dd if=/dev/rmt/0m of=x ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magtape. *Dd* is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary block sizes.

SEE ALSO

cp(1).

DIAGNOSTICS

f+p blocks in(out) numbers of full and partial blocks read(written)

BUGS

The ASCII/EBCDIC conversion tables are taken from the 256-character standard in the CACM Nov, 1968. The *ibm* conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

New-lines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

DELTA(1)

NAME

delta - make a delta (change) to an SCCS file

SYNOPSIS

delta [-rSID] [-s] [-n] [-glist] [-m[mrlist]] [-y[comment]] [-p] files

DESCRIPTION

Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

Delta makes a delta to each named SCCS file. If a directory is named, *delta* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read [see *WARNINGS*]; each line of the standard input is taken to be the name of an SCCS file to be processed.

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags [see *admin*(1)] that may be present in the SCCS file (see -m and -y keyletters below).

Keyletter arguments apply independently to each named file.

-rSID Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding *gets* for editing (*get -e*) on the same SCCS file were done by the same person (login name). The SID value specified with the -r keyletter can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command [see *get*(1)]. A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

-s Suppresses the issue on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

-n Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

-glist Specifies a *list* [see *get*(1) for the definition of *list*] of deltas which are to be *ignored* when the file is accessed at the change level (SID) created by this delta.

-m[mrlist] If the SCCS file has the v flag set [see *admin*(1)] then a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.

If -m is not used and the standard input is a terminal, the prompt MRs? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The MRs? prompt always precedes the comments? prompt (see -y keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the v flag has a value [see *admin*(1)], it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. If a non-zero exit status is returned from MR number validation program, *delta* terminates (it is assumed that

- the MR numbers were not all valid).
- y[comment]** Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.
If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.
- p** Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff(1)* format.

FILES

All files of the form *?-file* are explained in the *Source Code Control System User Guide*. The naming convention for these files is also described there.

- g-file** Existed before the execution of *delta*; removed after completion of *delta*.
- p-file** Existed before the execution of *delta*; may exist after completion of *delta*.
- q-file** Created during the execution of *delta*; removed after completion of *delta*.
- x-file** Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
- z-file** Created during the execution of *delta*; removed during the execution of *delta*.
- d-file** Created during the execution of *delta*; removed after completion of *delta*.
- /usr/bin/bdiff** Program to compute differences between the "gotten" file and the *g-file*.

WARNINGS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS [see *sccsfile(4)*] and will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get/delta* sequences should be used.

If the standard input (**-**) is specified on the *delta* command line, the **-m** (if necessary) and **-y** keyletters *must* also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.

SEE ALSO

admin(1), *bdiff(1)*, *cdc(1)*, *get(1)*, *help(1)*, *prs(1)*, *mdel(1)*, *sccsfile(4)* and "Source Code Control System User Guide" in the Software Development System manual.

DIAGNOSTICS

Use *help(1)* for explanations.

DEVNM(1M)

NAME

`devnm` - device name

SYNOPSIS

`/etc/devnm` [names]

DESCRIPTION

Devnm identifies the special file associated with the mounted file system where the argument *name* resides. (As a special case, both the block device name and the swap device name are printed for the argument name `/` if swapping is done on the same disk section as the **root** file system.) Argument names must be full path names.

This command is most commonly used by `/etc/rc` [see *brc(1M)*] to construct a mount table entry for the **root** device.

EXAMPLE

The command:

`/etc/devnm /usr`

produces

`dsk/0s1 /usr`

if `/usr` is mounted on `/dev/dsk/0s1`.

FILES

`/dev/dsk/*`

`/etc/mnttab`

SEE ALSO

`brc(1M)`, `setmnt(1M)`.

NAME

df - report number of free disk blocks

SYNOPSIS

df [*-t*] [*-f*] [*file-systems*]

DESCRIPTION

Df prints out the number of free blocks and free i-nodes available for on-line file systems by examining the counts kept in the super blocks; *file-systems* may be specified either by device name (e.g., */dev/dsk/0s1*) or by mounted directory name (e.g., */usr*). If the *file-systems* argument is unspecified, the free space on all of the mounted file systems is printed.

The *-t* flag causes the total allocated block figures to be reported as well.

If the *-f* flag is given, only an actual count of the blocks in the free list is made (free i-nodes are not reported). With this option, *df* will report on raw devices.

FILES

*/dev/dsk/**
/etc/mnttab

SEE ALSO

fs(4), *mnttab(4)*.

DIFF(1)

NAME

`diff` - differential file comparator

SYNOPSIS

`diff` [`-efbh`] *file1* *file2*

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is `-`, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

```
nl a n3,n4
nl,n2 d n3
nl,n2 c n3,n4
```

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging a for *d* and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs, where *nl* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by `<`, then all the lines that are affected in the second file flagged by `>`.

The `-b` option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The `-e` option produces a script of *a*, *c*, and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The `-f` option produces a similar script, not useful with *ed*, in the opposite order. In connection with `-e`, the following shell program may help maintain multiple versions of a file. Only an ancestral file (\$1) and a chain of version-to-version *ed* scripts (\$2,\$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option `-h` does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options `-e` and `-f` are unavailable with `-h`.

FILES

```
/tmp/d?????
/usr/lib/diffh for -b
```

SEE ALSO

`cmp`(1), `comm`(1), `ed`(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

BUGS

Editing scripts produced under the `-e` or `-f` option are naive about creating lines consisting of a single period (`.`).

WARNINGS

Missing newline at end of file X

indicates that the last line of file *X* did not have a new-line. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

NAME
diff3 - 3-way differential file comparison

SYNOPSIS
diff3 [-ex3] file1 file2 file3

DESCRIPTION
Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

```
====          all three files differ
=====1      file1 is different
=====2      file2 is different
=====3      file3 is different
```

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

```
f: n1 a      Text is to be appended after line number n1 in file f,
              where f = 1, 2, or 3.

f: n1 , n2 c  Text is to be changed in the range line n1 to line n2.
              If n1 = n2, the range may be abbreviated to n1.
```

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the **-e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged **====** and **=====3**. Option **-x (-3)** produces a script to incorporate only changes flagged **=====** (**=====3**). The following command will apply the resulting script to *file1*.

```
(cat script; echo '1,$p') | ed - file1
```

FILES
/tmp/d3*
/usr/lib/diff3prog

SEE ALSO
diff(1).

BUGS
Text lines that consist of a single , will defeat **-e**.
Files longer than 64K bytes will not work.

DIFFMK(1)

NAME

`diffmk` — mark differences between files

SYNOPSIS

`diffmk` *name1* *name2* *name3*

DESCRIPTION

Diffmk compares two versions of a file and creates a third file that includes “change mark” commands for *nroff* or *troff*(1). *Name1* and *name2* are the old and new versions of the file. *Diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter “change mark” (*.mc*) requests. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single ***.

If anyone is so inclined, *diffmk* can be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

where the file *macs* contains:

```
.pl 1  
.ll 77  
.nf  
.eo  
.nc
```

The *.ll* request might specify a different line length, depending on the nature of the program being printed. The *.eo* and *.nc* requests are probably needed only for C programs.

If the characters | and * are inappropriate, a copy of *diffmk* can be edited to change them (*diffmk* is a shell procedure).

SEE ALSO

`diff`(1).

“Nroff and Troff User Manual” in the Text Preparation System Manual.

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing *.sp* by *.sp 2* will produce a “change mark” on the preceding or following line of output.

NAME

`dircmp` - directory comparison

SYNOPSIS

`dircmp [-d] [-s] [-wn] dir1 dir2`

DESCRIPTION

Dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the file names common to both directories have the same contents.

- d Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff*(1).
- s Suppress messages about identical files.
- wn Change the width of the output line to *n* characters. The default width is 72.

SEE ALSO

`cmp`(1), `diff`(1).

DIS(1)

NAME

dis - 80286 disassembler

SYNOPSIS

dis [-o] [-V] [-L [-dsec] [-dasec] [-tsec] [-l string] files

DESCRIPTION

The *dis* command produces an assembly language listing of each of its object file arguments. The listing includes assembly statements and the binary that produced those statements.

The following options are interpreted by the disassembler and may be specified in any order.

- o Will print numbers in octal. Default is hexadecimal.
- L Invokes a lookup of C source labels in the symbol table for subsequent printing.
- dsec Disassembles the named section as data, printing the offset of the data from the beginning of the section.
- dasec Disassembles the named section as data, printing the actual address of the data.
- tsec Disassembles the named section as text.
- l string Will disassemble the library file specified as string. For example, one would issue the command *dis -l x -l z* to disassemble *libx.a* and *libz.a*. All libraries are assumed to be in */usr/lib*.

If the *-d*, *-da*, or *-t* options are specified, only those named sections from each user supplied file name will be disassembled. Otherwise, all sections containing text will be disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as [5], represents that the C breakpointable line number, starts with the following instruction. An expression such as <40> in the operand field, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A C function name will appear in the first column, followed by 0.

SEE ALSO

as(1), *cc(1)*, *ld(1)*.

DIAGNOSTICS

The self explanatory diagnostics indicate errors in the command line or problems encountered with the specific files.

NAME

divvy — divide disk allocation between file systems

SYNOPSIS

divvy

DESCRIPTION

The *divvy* utility facilitates the making of file systems on the System V/AT partition of both the primary and the secondary hard disk, and is usually invoked from the *installit* script after hard disk partitioning has been completed [see *fdisk(1M)*]; the utility can also be invoked independently.

Divvy prompts the user to accept default allocations or to re-organize the allocation of existing disk space between the */root*, */usr* and */tmp* file systems, and the system swap area. Allocations are given by hard disk unit number, 0 or 1, in quantities of 512-byte blocks. Total allocations for each hard disk unit are cross-checked against System V/AT partition size minus overhead tracks. When allocation has been accepted, *mkfs(1M)* is invoked to create the specified file systems.

SEE ALSO

fdisk(1M), *mkfs(1M)*, *showbad(1M)*.

This page intentionally left blank.

NAME

doscat - concatenate and print DOS files

SYNOPSIS

doscat [-bzS] [-D dev_dir] [dev:] file [files]

DESCRIPTION

Doscat reads each argument file in sequence and writes it on the standard output. If the specified file name is a DOS subdirectory, it will be skipped (with a message printed on the standard error---all messages will go to standard error, so only file contents should appear on the standard output). Rules for metacharacters, the DOS device prefix, and the default DOS device directory are the same as for *dosdir*(1).

- b Binary mode - print every byte up to the maximum file size listed in the DOS directory. Without this option operation is like DOS "Ascii mode"; i.e., all ctl-M, ctl-J (<CR><LF>) sequences will be mapped to ctl-J only, and printing will be terminated if a ctl-Z character is encountered (without printing the ctl-Z).
- z Ignore EOF - "Ascii mode" but ctl-Z will not terminate the print-out. This option has no effect if -b is specified as well.
- Dp Device directory changed to path p - same as *dosdir*(1).
- S Separator changed to "\ " - same as *dosdir*(1).

DOSCP(1)

NAME

`doscp` - copy files to and from DOS file systems

SYNOPSIS

`doscp` [-bfnzS] [-D dev_dir] [dev:] file [files] [dev:] target_dir

DESCRIPTION

Doscp is somewhere between DOS COPY and System V `cp(1)`. Unlike either of these commands, however, *doscp* does not rename files; i.e., there may be from 2, up to an arbitrary number of arguments, but in any case the last argument must specify a directory. Thus, except for the final path name, the tail (base) of each argument path name must specify a file, and this file name will be the name of the file copied into the target directory. File names may contain metacharacters which will be expanded according to System V rules if the direction of the copy is System V to DOS, and according to DOS rules if the direction of the copy is DOS to System V.

The direction of file copy is determined by the placement of the DOS device prefix (e.g. A:). The DOS prefix will only be recognized as such if it precedes either the first argument (file name) or the last argument (target directory name); there is no default DOS prefix. There must be at least one DOS prefix (no System V-to-System V copy) but no more than one DOS prefix (no DOS-to-DOS copy).

The rules for metacharacters and the default DOS device directory path are the same as for *dosdir(1)*. Examples:

- (a) `doscp A:asm test/d* /usr/tmp`
- (b) `doscp /etc/[p-r]* B:dhl`

The first line (a) above would cause all files with an "asm" extension, in the root directory, and all files beginning with the letter "d" in the subdirectory "test" to be copied from DOS device A to the local (System V) /usr/tmp directory. The second command (b) would copy all files beginning with either "p", "q" or "r" in the System-V /etc directory to the "dhl" subdirectory of DOS device B. Note that the meta characters in the second example are intended for expansion by the shell, `sh(1)`, and therefore the "[.]" non-DOS meta characters may be used. By the same token, the "*" in the first example risks shell expansion (if there were a subdirectory "test" in the System V current working directory), and might need to be quoted.

- b "Binary mode" - see *doscat(1)* - without this option copy terminates if a control-Z character is encountered (without copying the `ctl-Z`), and a copy from DOS maps all `ctl-M`, `ctl-J` (<CR><LF>) pairs to `ctl-J` only, while a copy to DOS performs the inverse mapping.
- f Ignore read-only attribute of DOS file - without this option, an attempt to overwrite a read-only DOS file will fail with an explanatory message. When a read-only DOS file is overwritten, the read-only attribute will remain. This option has no effect on a copy from DOS to System V. If the target System V file is read-only, a separate `clunod(1)` command will be required.
- m Preserve modification date of source file - otherwise the current System V time is used to stamp the copy.
- z Ignore EOF - allows Ascii mapping without control-Z termination. No effect if -b has been specified.
- Dp Device directory changed to path p - same as *dosdir(1)*.
- S Separator changed to "\ " - same as *dosdir(1)*.

NAME

`dosdir` - emulate "dir" command for DOS file systems

SYNOPSIS

`dosdir [-alvwS] [-C width] [-D dev_dir] [dev:] [path] [paths]`

DESCRIPTION

`Dosdir` provides a "DOS-like" listing for one or more files or directories specified in the argument list, differing from DOS in that it will process an arbitrary number of such arguments. The "*" and "?" metacharacters are handled according to DOS rules; i.e., they will not match a directory in the head of a path name, and if they match a directory in the tail (base) of path name, the match will result in a one-line listing of the directory in the display of its parent. Thus, the only way to invoke a full listing of the contents of a directory is to specify the exact name in the tail of a path name (except for the root directory, which will be listed by default if no arguments are given). The metacharacters are also DOS-like in that they are applied separately to the base and extension portions of a file-name (e.g., "name?com" will not match "name.com" — but it will, on the other hand, match "name|com.exe" using the DOS convention that a name with no extension is equivalent to "name.*", while, similarly, ".bat" is the same as "*.bat", and "name." is used to ask for a file "name" which actually has no extension). Of course, the metas are intended to match DOS files, but they can accidentally match System V files and be expanded by the shell. This can be avoided by placing any argument containing metas in double quotes:

```
dosdir "???" "/bin/*" (quotes very important here!)
dosdir A:f*.com f*.asm (not likely to match a local file)
```

The DOS device (drive) is specified by prefixing the first argument with the name of a System V file (presumably a special device file, though the image of a DOS file system might be placed in an ordinary file) followed by a colon, ":", as in DOS. The program expects to find this file in the directory /dev/dos, so DOS-like operation can be conveniently arranged by making links from the appropriate device files, and naming the links "A", "B", etc. For example:

```
mkdir /dev/dos
link /dev/dsk/fd /dev/dos/A
link /dev/dsk/fd048 /dev/dos/G
```

In general, linking from hard disk special name dev/rdisk/0sX, where X is replaced by 6, 7, 8, or 9, corresponds to DOS partitions 1 through 4. For example, DOS partition 1 corresponds to 0s6, partition 2 with 0s7, etc.

When no device prefix is found on the first argument, it defaults to "A"; a prefix will not be recognized on any other argument.

- a "Hidden" DOS files will appear in the listing of their parent directory; otherwise, a hidden file will only be listed if it is spelled out in the tail of a path name argument; a hidden file will not be matched by a metacharacter.
- l The normal DOS line for each file will be lengthened to include the file "attributes" which can be either or all of "rhs", where "r" is Read-Only, "h" is Hidden, and "s" is "System".
- v The first line of standard output will give the DOS OEM name and version.

DOSDIR(1)

- w Same as DOS; i.e., the DOS "dir" command defaults to a long listing, and this option gives a "wide" listing where only the file name and extension are shown in a multi-column display.
- Cn Change default display width (80 columns) to n columns; this option has no effect unless -w is also specified.
- Dp Change default device directory (/dev/dos) to path p.
- S Change default DOS path-name separator to "\". Since the back-slash character "\" must be escaped to be passed through the shell — sh(1) — command line, dosdir normally expects DOS path names to be separated by the forward slash "/", so they look the same as System V path names. If this option is used the command line might look like this:

```
dosdir -S B:\level1\level2\file
```

SEE ALSO

wn(7), fl(7) in the Runtime System manual.

NAME

`du` - summarize disk usage

SYNOPSIS

`du` [`-ars`] [`names`]

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, `.` is used.

The optional argument `-s` causes only the grand total (for each of the specified *names*) to be given. The optional argument `-a` causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

Du is normally silent about directories that cannot be read, files that cannot be opened, etc. The `-r` option will cause *du* to generate messages in such instances.

A file with two or more links is only counted once.

BUGS

If the `-a` option is not used, nondirectories given as arguments are not listed.

If there are too many distinct linked files, *du* will count the excess files more than once.

Files with holes in them will get an incorrect block count.

DUMP(1)

NAME

dump - dump selected parts of a common object file

SYNOPSIS

dump [-acfg~~h~~lorstV] [-z name] files

DESCRIPTION

The *dump* command dumps selected parts of each of its object *file* arguments.

This command will accept all object files in the common object file format. It will also accept archives of object files. It processes each file argument according to one or more of the following options:

- a Dump the archive header of each member of each archive file argument.
- c Dump the string table.
- f Dump each file header.
- g Dump the global symbols in the symbol table of a UNIX System V Release 2.0 archive.
- h Dump section headers.
- l Dump line number information.
- o Dump each optional header.
- r Dump relocation information.
- s Dump section contents.
- t Dump symbol table entries.
- z *name* Dump line number entries for the named function.
- V Output a message giving information about the version of dump being used.

The following *modifiers* are used in conjunction with the options listed above to modify their capabilities.

- d *number* Dump the section number or range of sections starting at *number* and ending either at the last section number or *number* specified by +d.
- +d *number* Dump sections in the range either beginning with first section or beginning with section specified by -d.
- n *name* Dump information pertaining only to the named entity. This *modifier* applies to -h, -s, -r, -l, and -t.
- p Suppress printing of the headers.
- t *index* Dump only the indexed symbol table entry. The -t used in conjunction with +t, specifies a range of symbol table entries.
- +t *index* Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the -t option.
- u Underline the name of the file for emphasis.
- v Dump information in symbolic representation rather than numeric (e.g., C_STATIC instead of 0X02). This *modifier* can be used with all the above options except -s and -o options of *dump*.

-zname,number

Dump line number entry or range of line numbers starting at *number* for the named function.

+znumber Dump line numbers starting at either function *name* or *number* specified by **-z**, up to *number* specified by **+z**.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the **-z** option may be replaced by a blank.

The **-z** and **-n** options that take a *name* modifier will only work with object files that contain debugging information.

The *dump* command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal, or decimal representation as appropriate.

SEE ALSO

a.out(4), ar(4) in the Software Development System manual.

ECHO(1)

NAME

echo - echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
\\	backslash
\n	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number <i>n</i> , which must start with a zero.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

sh(1).

NAME

ed, red - text editor

SYNOPSIS

ed [-] [file]

red [-] [file]

DESCRIPTION

Ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited. The optional - suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a *!shell command*. *Ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

Red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via *!shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec(4)* formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in *stty -tabs* or *stty tab3* mode [see *stty(1)*], the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

```
<:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10 and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

Ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character REs* match a *single* character:

- 1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.
- 1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:
 - a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([; see 1.4 below).

- b. (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).
 - c. \$ (currency symbol), which is special at the *end* of an entire RE (see 3.2 below).
 - d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the g command, below.)
- 1.3 A period (.) is a one-character RE that matches any character except new-line.
- 1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (-) may be used to indicate a range of consecutive ASCII characters; for example, [0-9] is equivalent to [0123456789]. The - loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., [])a-f] matches either a right square bracket (]) or one of the letters a through f inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct REs from one-character REs:

- 2.1 A one-character RE is a RE that matches whatever the one-character RE matches.
- 2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.
- 2.3 A one-character RE followed by \{m\}, \{m,\}, or \{m,n\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{m\} matches *exactly m* occurrences; \{m,\} matches *at least m* occurrences; \{m,n\} matches *any number* of occurrences *between m and n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.
- 2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.
- 2.5 A RE enclosed between the character sequences \(and\) is a RE that matches whatever the unadorned RE matches.
- 2.6 The expression \n matches the same string of characters as was matched by an expression enclosed between \(and\) *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression \(.*)\1\$ matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both):

- 3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.
- 3.2 A currency symbol (\$) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction *entire RE*\$ constrains the entire RE to match the entire line.

The null RE (e.g., */*) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character *.* addresses the current line.
2. The character *\$* addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. '*x*' addresses the line marked with the mark name character *x*, which must be a lowercase letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (*/*) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before *FILES* below.
6. A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign (+) or a minus sign (-) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; e.g., -5 is understood to mean *.-5*.
9. If an address ends with + or -, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address - refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character *.* in addresses is entirely equivalent to *-*.) Moreover, trailing + and - characters have a cumulative effect, so -- refers to the current line less 2.
10. For convenience, a comma (,) stands for the address pair *1,\$*, while a semicolon (;) stands for the pair *.,\$*.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6. above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by *l*, *n* or *p*, in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n* and *p* commands.

(.)a
<text>

The *append* command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command; it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.)c
<text>

The *change* command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(..)d

The *delete* command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e file

The *edit* command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by *!*, the rest of the line is taken to be a shell [*sh*(1)] command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E file

The *Edit* command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file*

If *file* is given, the *file-name* command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1,\$)g/*RE*/*command list*

In the global command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with *.* initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multiline list except the last line must be ended with a **; *a*, *i*, and *c* commands and associated input are permitted; the *.* terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before *FILES* below.

(1,\$)G/*RE*/

In the interactive Global command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, *.* is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an *&* causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *help* command gives a short error message that explains the reason for the most recent *?* diagnostic.

H

The *Help* command causes *ed* to enter a mode in which error messages are printed for all subsequent *?* diagnostics. It will also explain the previous *?* if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i

<*text*>

The insert command inserts the given text before the addressed line; *.* is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.,.+1)j

The join command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)kx

The mark command marks the addressed line with name *x*, which must be a lowercase letter. The address '*x*' then addresses this line; *.* is unchanged.

(..)l

The *list* command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab*, *backspace*) are represented by (hopefully) mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(..)ma

The *move* command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file; it is an error if address *a* falls within the range of moved lines; *.* is left at the last line moved.

(..)n

The *number* command prints the addressed lines, preceding each line by its line number and a tab character; *.* is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(..)p

The *print* command prints the addressed lines; *.* is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*; for example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a *** for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

The *quit* command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

(\$)r file

The *read* command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; *.* is set to the last line read in. If *file* is replaced by *!*, the rest of the line is taken to be a shell [*sh*(1)] command whose output is to be read. For example, "*\$r !ls*" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(..)s/RE/replacement/ or

(..)s/RE/replacement/g

The *substitute* command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator *g* appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of */* to delimit the RE and the *replacement*; *.* is left at the last line on which a substitution occurred. See also the last

paragraph before *FILES* below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters \n, where n is a digit, are replaced by the text matched by the n-th regular subexpression of the specified RE enclosed between \ (and \). When nested parenthesized subexpressions are present, n is determined by counting occurrences of \ (starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a g or v command list.

(.,)ta

This command acts just like the m command, except that a copy of the addressed lines is placed after address a (which may be 0); . is left at the last line of the copy.

u

The undo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent a, c, d, g, i, j, m, r, s, t, v, G, or V command.

(1,\$)v/RE/command list

This command is the same as the global command g except that the *command list* is executed with . initially set to every line that does not match the RE.

(1,\$)V/RE/

This command is the same as the interactive global command G except that the lines that are marked during the first step are those that do not match the RE.

(1,\$)w file

The write command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting [see *sh(1)*] dictates otherwise. The currently-remembered file name is not changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell [*sh(1)*] command whose standard input is the addressed lines. Such a shell command is not remembered as the current file name.

(\$)=

The line number of the addressed line is typed; . is unchanged by this command.

!shell command

The remainder of the line after the ! is sent to the UNIX shell [*sh(1)*] to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the

last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(.+1) <new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **+.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a ? and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If the closing delimiter of a RE or of a replacement string (e.g., **/**) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

FILES

/tmp/e# temporary; # is the process number.
ed.hup work is saved here if the terminal is hung up.

DIAGNOSTICS

? for command errors.
?file for an inaccessible file.
 (use the *help* and *Help* commands for detailed explanations).

If changes have been made in the buffer since the last **w** command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the **e** or **q** commands: it prints ? and allows one to continue editing. A second **e** or **q** command at this point will take effect. The **-** command-line option inhibits this feature.

SEE ALSO

grep(1), **sed(1)**, **sh(1)**, **stty(1)**, **fspec(4)**, **regex(5)**.
A Tutorial Introduction to the UNIX System Text Editor by B. W. Kernighan.
Advanced Editing on the UNIX System by B. W. Kernighan.

CAVEATS AND BUGS

A **!** command cannot be subject to a **g** or a **v** command.
 The **!** command and the **!** escape from the **e**, **r**, and **w** commands cannot be used if the editor is invoked from a restricted shell [see **sh(1)**].
 The sequence **\n** in a RE does not match a new-line character.
 The **/** command mishandles DEL.
 Characters are masked to 7 bits on input.

NAME

edit — text editor (variant of *ex* for casual users)

SYNOPSIS

edit [-r] name ...

DESCRIPTION

Edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command-oriented editor. The following brief introduction should help you get started with *edit*. If you are using a CRT terminal you may want to learn about the display editor *vi*.

BRIEF INTRODUCTION

To edit the contents of an existing file you begin with the command "*edit name*" to the shell. *Edit* makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run *edit* on it; you will cause an error diagnostic, but do not worry.

Edit prompts for commands with the character ':', which you should see after starting the editor. If you are editing an existing file, then you will have some lines in *edit's* buffer (its name for the copy of the file you are editing). Most commands to *edit* use its "current line" if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and hit carriage return (as you should after all *edit* commands) this current line will be printed. If you **delete** (**d**) the current line, *edit* will print the new current line. When you start editing, *edit* makes the last line of the file the current line. If you **delete** this last line, then the new last line becomes the current one. In general, after a **delete**, the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you give this command (typing a carriage return after the word **append**) *edit* will read lines from your terminal until you give a line consisting of just a ".", placing these lines after the current line. The last line you type then becomes the current line. The command **insert** (**i**) is like **append** but places the lines you give before, rather than after, the current line.

Edit numbers the lines in the buffer, with the first line having number 1. If you give the command "1" then *edit* will type this first line. If you then give the command **delete** *edit* will delete the first line, line 2 will become line 1, and *edit* will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command. You say "*s/old/new/*" where *old* is replaced by the old characters you want to get rid of and *new* is the new characters you want to replace it with.

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say "[Modified]" if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a **write** (**w**) command. You can then leave the editor by issuing a **quit** (**q**) command. If you run *edit* on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you will be warned that there has been "No write since last change" and *edit* will await another command. If you wish not to **write** the buffer out then you can issue another **quit** command. The buffer is then irretrievably discarded, and you return to the shell.

By using the **delete** and **append** commands, and giving line numbers to see lines in the file you can make any changes you desire. You should learn at least a few more things, however, if you are to use *edit* more than a few times.

The **change** (**c**) command will change the current line to a sequence of lines you supply (as in **append** you give lines up to a line consisting of only a "."). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e., "3,5change". You can print lines this way too. Thus "1,23p" prints the first 23 lines of the file.

The **undo** (**u**) command will reverse the effect of the last command you gave which changed the buffer. Thus if you give a **substitute** command which does not do what you want, you can say **undo** and the old contents of the line will be restored. You can also **undo** an **undo** command so that you can continue to change your mind. *Edit* will give you a warning message when commands you do affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an *undo* and looking to see what happened. If you decide that the change is ok, then you can *undo* again to get it back. Note that commands such as *write* and *quit* cannot be undone.

To look at the next line in the buffer you can just hit carriage return. To look at a number of lines hit "D (control key and, while it is held down D key, then let up both) rather than carriage return. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command "z.". The current line will then be the last line printed; you can get back to the line where you were before the "z." command by saying "z.". The z command can also be given other following characters "z-" prints a screen of text (or 24 lines) ending where you are; "z+" prints the next screenful. If you want less than a screenful of lines, type in "z.12" to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command "delete 5".

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backward and forward in the file for strings by giving commands of the form */text/* to search forward for *text* or *?text?* to search backward for *text*. If a search reaches the end of the file without finding the text it wraps, end around, and continues to search back to the line where you are. A useful feature here is a search of the form */text/* which searches for *text* at the beginning of a line. Similarly */text\$/* searches for *text* at the end of a line. You can leave off the trailing */* or *?* in these commands.

The current line has a symbolic name "."; this is most useful in a range of lines as in *.,\$print* which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name "\$". Thus the command *"\$ delete"* or *"\$d"* deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line *"\$-5"* is the fifth before the last, and *."+20"* is 20 lines after the present.

You can find out which line you are at by doing *.".=*. This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move you can then say *"10,20delete a"* which deletes these lines from the file and places them in a buffer named *a*. *Edit* has 26 such buffers named *a* through *z*. You can later get these lines back by doing *"put a"* to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files you can give an **edit** (**e**) command after copying the lines,

following it with the name of the other file you wish to edit, i.e., "edit chapter2". By changing *delete* to *yank* above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say "10,20move \$" for example. It is not necessary to use named buffers in this case (but you can if you wish).

SEE ALSO

ex(1), vi(1).

EFL(1)

NAME

efl -- Extended FORTRAN Language

SYNOPSIS

efl [options] [files]

DESCRIPTION

Efl compiles a program written in the EFL language into clean FORTRAN on the standard output. *Efl* provides the C-like control constructs of *ratfor* (1):

statement grouping with braces.

decision-making:

if, if-else, and select-case (also known as switch-case);
while, for, FORTRAN do, repeat, and repeat ... until loops;
multilevel break and next.

EFL has C-like data structures, e.g.:

```
struct
{
    integer flags(3)
    character(8) name
    long real coords(2)
} table(100)
```

The language offers generic functions, assignment operators (+ =, & =, etc.), and sequentially evaluated logical operators (& & and ||). There is a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) })
```

EFL also provides some syntactic "sugar":

free-form input:

multiple statements per line; automatic continuation; statement label names (not just numbers).

comments:

this is a comment.

translation of relational and logical operators:

>, >=, &, etc., become .GT., .GE., .AND., etc.

return expression to caller from function:

return (expression)

defines:

define name replacement

includes:

include file

Efl understands several option arguments: -w suppresses warning messages, -# suppresses comments in the generated program, and the default option -C causes comments to be included in the generated program.

An argument with an embedded = (equal sign) sets an EFL option as if it had appeared in an option statement at the start of the program. Many options are described in the reference manual. A set of defaults for a particular target machine may be selected by one of the choices: system=unix, system=geos, or system=cray. The default setting of the system option is the same as the machine the compiler is running on.

Other specific options determine the style of input/output, error handling, continuation conventions, the number of characters packed per word, and default formats.

Efl is best used with *f77(1)*.

SEE ALSO

cc(1), *f77(1)*, *ratfor(1)*.

ENABLE(1)

NAME

enable, disable — enable/disable LP printers

SYNOPSIS

enable printers

disable [**-c**] [**-r**[reason]] printers

DESCRIPTION

Enable activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

Disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

-c Cancel any requests that are currently printing on any of the designated printers.

-r[reason] Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next **-r** option. If the **-r** option is not present or the **-r** option is given without a reason, then a default reason will be used. *Reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), *lpstat*(1).

NAME

`env` - set environment for command execution

SYNOPSIS

`env [-] [name=value] ... [command args]`

DESCRIPTION

Env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The `-` flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

SEE ALSO

`sh(1)`.

`exec(2)`, `profile(4)`, `environ(5)` in the Software Development System manual.

ERRDEAD(1M)

NAME

errdead — extract error records from dump

SYNOPSIS

/etc/errdead *dumpfile* [*namelist*]

DESCRIPTION

When hardware errors are detected by the system, an error record that contains information pertinent to the error is generated. If the error-logging daemon *errdemon*(1M) is not active or if the system crashes before the record can be placed in the error file, the error information is held by the system in a local buffer. *Errdead* examines a system dump (or memory), extracts such error records, and passes them to *errpt*(1M) for analysis.

The *dumpfile* specifies the file (or memory) that is to be examined. The system name list is specified by *namelist*; if not given, */unix* is used.

FILES

<i>/unix</i>	system name list
<i>/usr/bin/errpt</i>	analysis program
<i>/usr/tmp/errXXXXXX</i>	temporary file

DIAGNOSTICS

Diagnostics may come from either *errdead* or *errpt*. In either case, they are intended to be self-explanatory.

SEE ALSO

errdemon(1M), *errpt*(1M).

NAME
errdemon — error-logging daemon

SYNOPSIS
/usr/lib/errdemon [file]

DESCRIPTION
The error logging daemon *errdemon* collects error records from the operating system by reading the special file */dev/error* and places them in *file*. If *file* is not specified when the daemon is activated, */usr/adm/errfile* is used. Note that *file* is created if it does not exist; otherwise, error records are appended to it, so that no previous error data is lost. No analysis of the error records is done by *errdemon*; that responsibility is left to *errpt(1M)*. The error-logging daemon is terminated by sending it a software kill signal [see *kill(1)*]. Only the super-user may start the daemon, and only one daemon may be active at any time.

FILES
/dev/error source of error records
/usr/adm/errfile repository for error records

DIAGNOSTICS
The diagnostics produced by *errdemon* are intended to be self-explanatory.

SEE ALSO
errpt(1M), *errstop(1M)*, *kill(1)*, *err(7)*.

ERRPT(1M)

NAME

`errpt` — process a report of logged errors

SYNOPSIS

`errpt` [options] [files]

DESCRIPTION

`Errpt` processes data collected by the error logging mechanism [`errdemon(1M)`] and generates a report of that data. The default report is a summary of all errors posted in the files named. Options apply to all files and are described below. If no files are specified, `errpt` attempts to use `/usr/adm/errfile` as *file*.

A summary report notes the options that may limit its completeness, records the time stamped on the earliest and latest errors encountered, and gives the total number of errors of one or more types. Each device summary contains the total number of unrecovered errors, recovered errors, errors unable to be logged, I/O operations on the device, and miscellaneous activities that occurred on the device. The number of times that `errpt` has difficulty reading input data is included as read errors.

Any detailed report contains, in addition to specific error information, all instances of the error logging process being started and stopped, and any time changes [via `date(1)`] that took place during the interval being processed. A summary of each error type included in the report is appended to a detailed report.

A report may be limited to certain records in the following ways:

- `-s date` Ignore all records posted earlier than *date*, where *date* has the form *mmddhhmmyy*, consistent in meaning with the `date(1)` command.
- `-e date` Ignore all records posted later than *date*, whose form is as described above.
- `-a` Produce a detailed report that includes all error types.
- `-d devlist` A detailed report is limited to data about devices given in *devlist*, where *devlist* can be one of two forms: a list of device identifiers separated from one another by a comma, or a list of device identifiers enclosed in double quotes and separated from one another by a comma and/or more spaces. `Errpt` is familiar with the common form of identifiers (e.g., `rs03`, `RS04`, `hs`; see Section 7 of this volume). For the 3B20 computer the devices for which errors are logged are `DFC`, `IOP`, and `MT`. For Digital Equipment Corporation machines, the (block) devices for which errors are logged are `RP03`, `RP04`, `RP05`, `RP06`, `RP07`, `RS03`, `RS04`, `TS11`, `TUI0`, `TUI6`, `TU78`, `RK05`, `RK06`, `RK07`, `RM05`, `RM80`, and `RF11`. Additional identifiers are `int` and `mem` which include detailed reports of stray-interrupt and memory-parity type errors, respectively.
- `-p n` Limit the size of a detailed report to *n* pages.
- `-f` In a detailed report, limit the reporting of block device errors to unrecovered errors.

FILES

`/usr/adm/errfile` default error file

SEE ALSO

`date(1)`, `errdead(1M)`, `errdemon(1M)`, `errfile(4)`.

NAME

errstop - terminate the error-logging daemon

SYNOPSIS

/etc/errstop [*namelist*]

DESCRIPTION

The error-logging daemon *errdemon*(1M) is terminated by using *errstop*. This is accomplished by executing *ps*(1) to determine the daemon's identity and then sending it a software kill signal [see *signal*(2)]; */unix* is used as the system *namelist* if none is specified. Only the super-user may use *errstop*.

FILES

/unix default system *namelist*

DIAGNOSTICS

The diagnostics produced by *errstop* are intended to be self-explanatory.

SEE ALSO

errdemon(1M), *ps*(1), *kill*(2), *signal*(2).

EX(1)

NAME

ex - text editor

SYNOPSIS

ex [-] [-v] [-t tag] [-r] [-R] [+command] [-l] name
...

DESCRIPTION

Ex is the root of a family of editors: *ex* and *vi*. *Ex* is a superset of *ed*, with the most notable extension being a display editing facility. Display based editing is the focus of *vi*.

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(1), which is a command which focuses on the display editing portion of *ex*.

DOCUMENTATION

The *Ex Reference Manual* is a comprehensive and complete manual for the command mode features of *ex*, but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of *ex* see the editing documents written by Brian Kernighan for the editor *ed*; the material in the introductory and advanced documents works also with *ex*.

An Introduction to Display Editing with Vi introduces the display editor *vi* and provides reference material on *vi*. The *Vi Quick Reference* card summarizes the commands of *vi* in a useful, functional way, and is useful with the *Introduction*. The *vi*(1) manual page can also be used as reference.

FOR ED USERS

If you have used *ed* you will find that *ex* has a number of new features useful on CRT terminals. Intelligent terminals and high-speed terminals are very pleasant to use with *vi*. Generally, the editor uses far more of the capabilities of terminals than *ed* does, and uses the terminal capability data base *terminfo*(4) and the type of the terminal you are using from the variable *TERM* in the environment to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using *vi*(1).

Ex contains a number of new features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Hitting **D** causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just hitting return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

Ex gives you more help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. *Ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files unless you edited them so that you do not accidentally clobber with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the phone, you can use the editor **recover** command to retrieve your work. This will get you back to within a few lines of where you left off.

Ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The

metacharacter '%' is also available in forming file names and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

There is a command **&** in *ex* which repeats the last **substitute** command. In addition there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. *Ex* also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

Ex has a set of *options* which you can set to tailor it to your liking. One option which is very useful is the *autoindent* option which allows the editor to automatically supply leading white space to align text. You can then use the **D** key as a backtab and space and tab forward to align new code easily.

Miscellaneous new useful features include an intelligent **join (J)** command which supplies white space between joined lines automatically, commands **<** and **>** which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort*.

INVOCATION OPTIONS

The following invocation options are interpreted by *ex*:

- Suppress all interactive-user feedback. This is useful in processing editor scripts.
- v Invokes *vi*
- t *tagfR* Edit the file containing the *tag* and position the editor at its definition.
- r *file* Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
- R *Readonly* mode set prevents accidentally overwriting the file.
- +*command* Begin editing by executing the specified editor search or positioning *command*.
- l **LISP** mode; indents appropriately for lisp code, the **O** **]** **||** and **||** commands in *vi* are modified to have meaning for *lisp*.

The *name* argument indicates files to be edited.

Ex States

- | | |
|---------|--|
| Command | Normal and initial state. Input prompted for by : . Your kill character cancels partial command. |
| Insert | Entered by a i and c . Arbitrary text may be entered. Insert is normally terminated by line having only . on it, or abnormally with an interrupt. |
| Visual | Entered by vi , terminates with Q or ^\ . |

EX(1)

Ex command names and abbreviations

abbrev	ab	next	n	unabbrev	una
append	a	number	nu	undo	u
args	ar			unmap	unm
change	c	preserve	pre	version	ve
copy	co	print	p	visual	vi
delete	d	put	pu	write	w
edit	e	quit	q	xit	x
file	f	read	re	yank	ya
global	g	recover	rec	window	z
insert	i	rewind	rew	escape	!
join	j	set	se	lshift	<
list	l	shell	sh	print next	CR
map		source	so	resubst	&
mark	ma	stop	st	rshift	>
move	m	substitute	s	scroll	^D

Ex Command Addresses

<i>n</i>	line <i>n</i>	<i>/pat</i>	next with <i>pat</i>
.	current	<i>?pat</i>	previous with <i>pat</i>
\$	last	<i>x-n</i>	<i>n</i> before <i>x</i>
+	next	<i>x,y</i>	<i>x</i> through <i>y</i>
-	previous	<i>^x</i>	marked with <i>x</i>
+ <i>n</i>	<i>n</i> forward	<i>''</i>	previous context
%	1,\$		

Initializing options

EXINIT	place set's here in environment var.
\$HOME/.exrc	editor initialization file
./exrc	editor initialization file
set <i>x</i>	enable option
set no <i>x</i>	disable option
set <i>x=val</i>	give value <i>val</i>
set	show changed options
set all	show all options
set <i>x?</i>	show value of option <i>x</i>

Most useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
lisp		() { } are s-exp's
list		print ^I for tab, \$ at end
magic		. [* special in patterns
number	nu	number lines
paragraphs	para	macro names which start ...
redraw		simulate smart terminal
scroll		command mode lines
sections	sect	macro names ...
shiftwidth	sw	for < >, and input ^D
showmatch	sm	to) and } as typed
showmode	smd	show insert mode in <i>vi</i>
slowopen	slow	stop updates during insert
window		visual mode lines
wrapscan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

Scanning pattern formation

.	beginning of line
\$	end of line
.	any character
\<	beginning of word
\>	end of word
[<i>str</i>]	any char in <i>str</i>
{ <i>str</i> }	... not in <i>str</i>
[<i>x-y</i>]	... between <i>x</i> and <i>y</i>
*	any number of preceding

AUTHOR

vi and *ex* are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/usr/lib/terminfo	describes capabilities of terminals
\$HOME/.exrc	editor startup file
./exrc	editor startup file
/tmp/Exnnnnn	editor temporary
/tmp/Rxnnnnn	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), cd(1), edit(1), grep(1), sed(1), vi(1).
 curses(3X), term(4), terminfo(4) in the Software Development System manual.

CAVEATS AND BUGS

The version of *ex* that runs on the PDP-11 does not support the full command set due to space limitations. The commands which are not supported are detailed in the "Ex Reference Manual." The most notable commands which are missing are the macro and abbreviation facilities.

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screenful of output may result if long lines are present.

File input/output errors do not print a name if the command line '-?' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

EXPR(1)

NAME

`expr` — evaluate arguments as an expression

SYNOPSIS

`expr` arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that 0 is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2s complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within {} symbols.

`expr \{ expr`

returns the first `expr` if it is neither null nor 0, otherwise returns the second `expr`.

`expr \& expr`

returns the first `expr` if neither `expr` is null or 0, otherwise returns 0.

`expr { =, >, >=, <, <=, != } expr`

returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

`expr { +, - } expr`

addition or subtraction of integer-valued arguments.

`expr { *, /, % } expr`

multiplication, division, or remainder of the integer-valued arguments.

`expr : expr`

The matching operator `:` compares the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of `ed(1)`, except that all patterns are "anchored" (i.e., begin with `^`) and, therefore, `^` is not a special character, in that context. Normally, the matching operator returns the number of characters matched (0 on failure). Alternatively, the `\(...\)` pattern symbols can be used to return a portion of the first argument.

EXAMPLES

1. `a=expr $a + 1`

adds 1 to the shell variable `a`.

2. `# For $a equal to either "/usr/abc/file" or just "file"`

`expr $a : .*^(.*) \| $a`

returns the last segment of a path name (i.e., file). Watch out for `/` alone as an argument: `expr` will take it as the division operator (see BUGS below).

3. `# A better representation of example 2.`

`expr // $a : .*^(.*)`

The addition of the `//` characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. `expr $VAR : .*`

returns the number of characters in `$VAR`.

SEE ALSO

ed(1), sh(1).

EXIT CODEAs a side effect of expression evaluation, *expr* returns the following exit values:

- 0 if the expression is neither null nor 0
- 1 if the expression is null or 0
- 2 for invalid expressions.

DIAGNOSTICS

syntax error for operator/operand errors
non-numeric argument if arithmetic is attempted on such a string

BUGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If *\$a* is an =, the command:

```
expr $a = =
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

```
expr X$a = X=
```

NAME

f77 - FORTRAN 77 compiler

SYNOPSIS

f77 [options] files

DESCRIPTION

F77 is the UNIX System FORTRAN 77 compiler; it accepts several types of *file* arguments:

Arguments whose names end with *.f* are taken to be FORTRAN 77 source programs; they are compiled and each object program is left in the current directory in a file whose name is that of the source, with *.o* substituted for *.f*.

Arguments whose names end with *.r* or *.e* are taken to be RATFOR or EFL source programs, respectively. These are first transformed by the appropriate preprocessor, then compiled by *f77*, producing *.o* files.

In the same way, arguments whose names end with *.c* or *.s* are taken to be C or assembly source programs and are compiled or assembled, producing *.o* files.

The following *options* have the same meaning as in *cc(1)* [see *ld(1)* for link editor options]:

- c Suppress link editing and produce *.o* files for each source file.
- p Prepare object files for profiling [see *prof(1)*].
- O Invoke an object-code optimizer.
- S Compile the named programs and leave the assembler-language output in corresponding files whose names are suffixed with *.s*. (No *.o* files are created.)
- o*output* Name the final output file *output*, instead of *a.out*.
- g Generate additional information needed for the use of *sdb(1)*.

The following *options* are peculiar to *f77*:

- onetrip Compile DO loops that are performed at least once if reached. (FORTRAN 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)
- 1 Same as **-onetrip**.
- 66 Suppress extensions which enhance FORTRAN 66 compatibility.
- C Generate code for run-time subscript range-checking.
- U Do not "fold" cases. *F77* is normally a no-case language (i.e., *a* is equal to *A*). The **-U** option causes *f77* to treat upper and lower cases to be separate.
- u Make the default type of a variable *undefined*, rather than using the default FORTRAN rules.
- v **Verbose** mode. Provide diagnostics for each process during compilation.
- w Suppress all warning messages. If the option is **-w66**, only FORTRAN 66 compatibility warnings are suppressed.
- F Apply EFL and RATFOR preprocessor to relevant files, put the result in files whose names have their suffix changed to *.f*. (No *.o* files are created.)
- m Apply the M4 preprocessor to each EFL or RATFOR source file before transforming with the *ratfor(1)* or *efl(1)* processors.
- E The remaining characters in the argument are used as an EFL flag argument whenever processing a *.e* file.
- R The remaining characters in the argument are used as a RATFOR flag argument whenever processing a *.r* file.

Other arguments are taken to be either link-editor option arguments or *f77*-compatible object programs (typically produced by an earlier run), or libraries of *f77*-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the default name **a.out**.

FILES

<code>file.[f]resc</code>	input file
<code>file.o</code>	object file
<code>a.out</code>	linked output
<code>./fort[<i>pid</i>].?</code>	temporary
<code>/usr/lib/f77pass1</code>	compiler
<code>/usr/lib/f77pass2</code>	pass 2
<code>/lib/c2</code>	optional optimizer
<code>/usr/lib/<model>/libF77.a</code>	intrinsic function library
<code>/usr/lib/<model>/libI77.a</code>	FORTRAN I/O library
<code>/lib/<model>/libc.a</code>	C library; see Section 3 of this Manual.

where "<model>" is small, large, or huge.

SEE ALSO

`asa(1)`, `cc(1)`, `efl(1)`, `fsplit(1)`, `ld(1)`, `m4(1)`, `prof(1)`, `ratfor(1)`, `sdb(1)`.

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the link editor `ld(1)`.

FACTOR(1)

NAME

factor — factor a number

SYNOPSIS

factor [number]

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{36} (about 7.2×10^{10}) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to \sqrt{n} and occurs when n is prime or the square of a prime. It takes 1 minute to factor a prime near 10^{14} on a PDP-11.

DIAGNOSTICS

“Ouch” for input out of range or for garbage input.

NAME

fdisk—fixed disk utility

SYNOPSIS

fdisk

DESCRIPTION

Fdisk initializes a DOS or System V/AT partition on either the primary or the secondary hard disk. The initialization process takes two steps to complete: assignment of partition and bad track mapping.

The *fdisk* partition scheme is compatible with DOS, and it is possible for a DOS partition to co-exist on the hard disk with System V/AT. After partition initialization, file systems must be made on the hard disk partition. The *divvy(1M)* utility is provided to assist in this process.

The *fdisk* utility is menu driven with six options;

1. Create a partition
2. Change Active partition
3. Delete a partition
4. Display partition information
5. Bad track mapping
6. Advance current disk to next

SEE ALSO

divvy(1M), *mkfs(1M)*, *setup(1M)*, *showbad(1M)*.

FF(1M)

NAME

`ff` — list file names and statistics for a file system

SYNOPSIS

`/etc/ff` [*options*] *special*

DESCRIPTION

Ff reads the i-list and directories of the *special* file, assuming it to be a file system, saving i-node data for files which match the selection criteria. Output consists of the path name for each saved i-node, plus any other file information requested using the print *options* below. Output fields are positional. The output is produced in i-node order; fields are separated by tabs. The default line produced by *ff* is:

path-name i-number

With all *options* enabled, output fields would be:

path-name i-number size uid

The argument *n* in the *option* descriptions that follow is used as a decimal integer (optionally signed), where *+n* means more than *n*, *-n* means less than, *n*, and *n* means exactly *n*. A day is defined as a 24-hour period.

- `-I` Do not print the i-node number after each path name.
- `-I` Generate a supplementary list of all path names for multiply linked files.
- `-p~prefix` The specified *prefix* will be added to each generated path name. The default is ..
- `-s` Print the file size, in bytes, after each path name.
- `-u` Print the owner's login name after each path name.
- `-a~n` Select if the i-node has been accessed in *n* days.
- `-m~n` Select if the i-node has been modified in *n* days.
- `-c~n` Select if the i-node has been changed in *n* days.
- `-n~file` Select if the i-node has been modified more recently than the argument *file*.
- `-i~i-node-list` Generate names for only those i-nodes specified in *i-node-list*.

EXAMPLES

To generate a list of the names of all files on a specified file system:

```
ff -I /dev/diskroot
```

To produce an index of files and i-numbers which are on a file system and have been modified in the last 24 hours:

```
ff -m -l /dev/diskusr > /log/incbackup/usr/tuesday
```

To obtain the path names for i-nodes 451 and 76 on a specified file system:

```
ff -i 451,76 /dev/rdisk0s7
```

SEE ALSO

`finc(1M)`, `find(1)`, `frec(1M)`, `ncheck(1M)`.

BUGS

Only a single path name out of any possible ones will be generated for a multiply linked i-node, unless the `-l` option is specified. When `-l` is specified, no selection criteria apply to the names generated. All possible names for every linked file on the file system will be included in the output.

On very large file systems, memory may run out before `ff` does.

FILE(1)

NAME

`file` — determine file type

SYNOPSIS

`file [-c] [-f ffile] [-m mfile] arg ...`

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable *a.out*, *file* will print the version stamp, provided it is greater than 0 (see *ld(1)*).

If the `-f` option is given, the next argument is taken to be a file containing the names of the files to be examined.

File uses the file `/etc/magic` to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of `/etc/magic` explains its format.

The `-m` option instructs *file* to use an alternate magic file.

The `-c` flag causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under `-c`.

SEE ALSO

ld(1).

NAME filesave - daily/weekly UNIX system file system backup

SYNOPSIS
/etc/filesave.?

DESCRIPTION

This shell script is provided as a model. It is designed to provide a simple, interactive operator environment for file backup. *Filesave.?* is for daily disk-to-disk backup.

The suffix *.?* can be used to name another system where two (or more) machines share disk drives and one or the other of the systems is used to perform backup on both.

SEE ALSO
shutdown(1M), volcopy(1M).

FINC(1M)

NAME

finc — fast incremental backup

SYNOPSIS

finc [*selection-criteria*] *file-system raw-device*

DESCRIPTION

Finc selectively copies the input *file-system* to the output *raw-device*. The media must be formatted as a backup and labeled by *labelit* [see *format*(1M) in this manual]. The cautious will want to mount the input *file-system* read-only to insure an accurate backup, although acceptable results can be obtained in read-write mode. The selection is controlled by the *selection-criteria*, accepting only those i-nodes/files for whom the conditions are true.

It is recommended that production of the *finc* media be preceded by the *ff* command, and the output of *ff* be saved as an index of the media's contents. Files on a *finc* media may be recovered with the *frec* command.

The argument *n* in the *selection-criteria* which follow is used as a decimal integer (optionally signed), where *+n* means more than *n*, *-n* means less than *n*, and *n* means exactly *n*. A day is defined as 24 hours.

- a n* True if the file has been accessed in *n* days.
- m n* True if the file has been modified in *n* days.
- c n* True if the i-node has been changed in *n* days.
- n/file* True for any file which has been modified more recently than the argument *file*.

EXAMPLES

To write a floppy disk consisting of all files from *file-system /usr* modified in the last 48 hours:

```
finc -m -2 /dev/rdisk/0s7 /dev/rdisk/0s24
```

SEE ALSO

cpio(1), *ff*(1M), *frec*(1M), *volcopy*(1M).

NAME

find — find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each path name in the *path-name-list* (i.e., one or more path names) seeking files that match a Boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where $+n$ means more than *n*, $-n$ means less than *n* and *n* means exactly *n*.

- name file** True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for `[`, `?` and `*`).
- perm onum** True if the file permission flags exactly match the octal number *onum* [see *chmod*(1)]. If *onum* is prefixed by a minus sign, more flag bits [017777, see *stat*(2)] become significant and the flags are compared.
- type c** True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **p**, or **f** for block special file, character special file, directory, fifo (a.k.a named pipe), or plain file respectively.
- links n** True if the file has *n* links.
- user uname** True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the */etc/passwd* file, it is taken as a user ID.
- group gname** True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the */etc/group* file, it is taken as a group ID.
- size n[c]** True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a *c*, the size is in characters.
- atime n** True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by *find* itself.
- mtime n** True if the file has been modified in *n* days.
- ctime n** True if the file has been changed in *n* days.
- exec cmd** True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semi-colon. A command argument `{` is replaced by the current path name.
- ok cmd** Like **-exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing *y*.
- print** Always true; causes the current path name to be printed.
- cpio device** Always true; write the current file on *device* in *cpio* (4) format (5120-byte records).
- newer file** True if the current file has been modified more recently than the argument *file*.
- depth** Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio* (1) to transfer files that are contained in directories without write permission.

FIND(1)

(*expression*) True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

- 1) The negation of a primary (! is the unary *not* operator).
- 2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
- 3) Alternation of primaries (-o is the *or* operator).

EXAMPLE

To remove all files named **a.out** or ***.o** that have not been accessed for a week:

```
find / \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

FILES

```
/etc/passwd  
/etc/group
```

SEE ALSO

chmod(1), cpio(1), sh(1), test(1).
stat(2), cpio(4), fs(4) in the Software Development System manual.

NAME
format - format floppy and hard disk tracks

SYNOPSIS
/etc/format {-f first} {-l last} {-i interleave} {-s device}

DESCRIPTION
Format formats floppy and Winchester hard disks. Unless otherwise specified, formatting starts at track 0 and continues until an error is returned at the end of a partition.

The -f and -l options specify the first and last cylinder to be formatted. The default interleave of 3 may be changed by using the -i option for hard disks only. There is no interleave option possible for formats of floppy disks.

The -s option will make the hard disk bootable. It creates the master boot block, and is similar to the DOS format command options '/S/V'. This option is only useful for hard disks.

Device must specify a raw (character) device.

EXAMPLES

<code>format -s /dev/rdisk/0s10</code>	formats the entire disk, and makes it bootable.
<code>format /dev/rdisk/fd096</code>	formats a 96 tpi floppy.

FILES
*/dev/rdisk/** raw device for partition to be formatted

FREC(1M)

NAME

`frec` — recover files from a backup device

SYNOPSIS

```
/etc/frec [ -p path ] [ -f reqfile ] raw-device i-number:name ...
```

DESCRIPTION

`Frec` recovers files from the specified *raw-device* backup written by `volcopy(1M)` or `finc(1M)`, given their *i-numbers*. The data for each recovery request will be written into the file given by *name*.

The `-p` option allows you to specify a default prefixing *path* different from your current working directory. This will be prefixed to any *names* that are not fully qualified, i.e., that do not begin with `/` or `./`. If any directories are missing in the paths of recovery *names* they will be created.

- `-p path` Specifies a prefixing *path* to be used to fully qualify any names that do not start with `/` or `./`.
- `-f reqfile` Specifies a file which contains recovery requests. The format is *i-number:newname*, one per line.

EXAMPLES

To recover a file, *i-number* 1216 when backed-up, into a file named `junk` in your current working directory:

```
frec /dev/rdisk/0s24 1216:junk
```

To recover files with *i-numbers* 14156, 1232, and 3141 into files `/usr/src/cmd/a`, `/usr/src/cmd/b` and `/usr/joe/a.c`:

```
frec -p /usr/src/cmd /dev/rdisk/0s24 14156:a 1232:b  
3141:/usr/joe/a.c
```

SEE ALSO

`cpio(1)`, `ff(1M)`, `finc(1M)`, `volcopy(1M)`.

BUGS

While paving a path (i.e., creating the intermediate directories contained in a path name) `frec` can only recover *i-node* fields for those directories contained on the device and requested for recovery.

NAME

fsck, *dfscck* - file system consistency check and interactive repair

SYNOPSIS

```
/etc/fsck [-y] [-n] [-sX] [-SX] [-t file] [-q] [-D] [-f] [-b]
[ file-systems ]
```

```
/etc/dfscck [ options1 ] filsys1 ... - [ options2 ] filsys2 ...
```

DESCRIPTION

Fsck

Fsck audits and interactively repairs inconsistent conditions for UNIX system files. If the file system is consistent, then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent, the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *fsck* will default to a **-n** action.

Fsck has more consistency checks than its predecessors *check*, *dcheck*, *fcheck*, and *icheck* combined.

The following options are interpreted by *fsck*.

- y** Assume a yes response to all questions asked by *fsck*.
- n** Assume a no response to all questions asked by *fsck*; do not open the file system for writing.
- sX** Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the superblock of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used or written on the file system.

The **-sX** option allows for creating an optimal free-list organization. The following forms of *X* are supported for the following devices:

- s3** (RP03)
- s4** (RP04, RP05, RP06)
- sBlocks-per-cylinder:Blocks-to-skip** (for anything else)

If *X* is not given, the values used when the file system was created are used. If these values were not specified, then the value **400:7** is used.

- SX** Conditionally reconstruct the free list. This option is like **-sX** above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **-S** will force a no response to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.
- t** If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **-t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **-t** flag, *fsck* will prompt the operator for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

FSCK(1M)

- q Quiet *fsck*. Do not print size-check messages in Phase 1. Unreferenced **fifo**s will silently be removed. If *fsck* requires it, counts in the superblock will be automatically fixed and the free list salvaged.
- D Directories are checked for bad blocks. Useful after system crashes.
- f Fast check. Check block and sizes (Phase 1) and check the free list (Phase 5). The free list will be reconstructed (Phase 6) if it is necessary.
- b Reboot. If the file system being checked is the root file system and modifications have been made, then either remount the root file system or reboot the machine. A remount is done only if there was minor damage.

If no *file-systems* are specified, *fsck* will read a list of default file systems from the file */etc/checklist*.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
 - Incorrect number of blocks.
 - Directory size not 16-byte aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
 - File pointing to unallocated i-node.
 - I-node number out of range.
8. Superblock checks:
 - More than 65536 i-nodes.
 - More blocks for i-nodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. If it is empty, *fsck* will silently remove them. *Fsck* will force the reconnection of nonempty directories. The name assigned is the i-node number. The only restriction is that the directory **lost+found** must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster and should be used with everything but the *root* file system.

Dfsck

Dfsck allows two file system checks on two different drives simultaneously. *options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A - is the separator between the file system groups.

The *dfsck* program permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsck* will print the file system name for each message to the operator. When answering a question from *dfsck*, the operator must prefix the response with a 1 or a 2 (indicating that the

answer refers to the first or second file system group).

Do not use *dfsck* to check the *root* file system.

FILES

/etc/checklist contains default list of file systems to check.
/etc/checkall optimizing *dfsck* shell file.

SEE ALSO

checkall(1M), *clri(1M)*, *ncheck(1M)*, *uadmin(2)*, *checklist(4)*, *fs(4)*, *crash(1M)*.

Administrative Advice in the Runtime System manual.

BUGS

I-node numbers for *.* and *..* in each directory should be checked for validity.

DIAGNOSTICS

The diagnostics produced by *fsck* are intended to be self-explanatory.

FSDB(1M)

NAME

fsdb — file system debugger

SYNOPSIS

/etc/fsdb special [-]

DESCRIPTION

Fsdb can be used to patch up a damaged file system after a crash. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the file system tree.

Fsdb contains several error-checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking *fsdb* with the optional - argument or by the use of the O symbol. (*Fsdb* reads the i-size and f-size entries from the super block of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

Fsdb reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by *fsdb* are:

#	absolute address
i	convert from i-number to i-node address
b	convert to block address
d	directory slot offset
+, -	address arithmetic
q	quit
>, <	save, restore an address
=	numerical assignment
= +	incremental assignment
= -	decremental assignment
= "	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the p symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words
e	print as decimal words

c print as characters
b print as octal bytes

The **f** symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs, and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. I-nodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a#	data block numbers (0 -- 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i prints i-number 386 in an i-node format. This now becomes the current working i-node.

ln=4 changes the link count for the working i-node to 4.

ln=+1 increments the link count by 1.

fc prints, in ASCII, block zero of the file associated with the working i-node.

2i.fd prints the first 32 directory entries for the root i-node of this file system.

d5i.fc changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.

512B.p0o prints the super block of this file system in octal.

2i.a0b.d7=3 changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.

d7.nm="name" changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.

FSSTAT(1M)

NAME

fsstat - file system status

SYNOPSIS

/etc/fsstat file-system

DESCRIPTION

Fsstat reports on the status of *file-system*. During startup, this command is used to decide if the file system needs checking before it is mounted. It succeeds if the file system is unmounted and appears okay. For the root file system, it succeeds if it is active and not marked bad.

SEE ALSO

fs(4) in the Software Development System manual.

DIAGNOSTICS

If successful, the command has an exit status of 0. Otherwise, the command has an exit status of 1 if the file system needs to be checked, 2 if mounted, and 3 for other failures.

NAME

fuser - identify processes using a file or file structure

SYNOPSIS

`/etc/fuser [-ku] files [-] [[-ku] files]`

DESCRIPTION

Fuser lists the process IDs of the processes using the *files* specified as arguments. For block special devices, all processes using any file on that device are listed. The process ID is followed by **c**, **p** or **r** if the process is using the file as its current directory, the parent of its current directory (only when in use by the system), or its root directory, respectively. If the **-u** option is specified, the login name, in parentheses, also follows the process ID. In addition, if the **-k** option is specified, the **SIGKILL** signal is sent to each process. Only the super-user can terminate another user's process [see *kill(2)*]. Options may be respecified between groups of files. The new set of options replaces the old set, with a lone dash canceling any options currently in force.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

EXAMPLES

`fuser -ku /dev/dsk/1s?`

will terminate all processes that are preventing disk drive one from being unmounted if typed by the super-user, listing the process ID and login name of each as it is killed.

`fuser -u /etc/passwd`

will list process IDs and login names of processes that have the password file open.

`fuser -ku /dev/dsk/1s? -u /etc/passwd`

will do both of the above examples in a single command line.

FILES

<code>/unix</code>	for name list
<code>/dev/kmem</code>	for system image
<code>/dev/mem</code>	also for system image

SEE ALSO

`mount(1M)`, `ps(1)`, `kill(2)`, `signal(2)`.

This page intentionally left blank.

NAME

fwtmp, *wtmpfix* — manipulate connect accounting records

SYNOPSIS

```
/usr/lib/acct/fwtmp [-ic]
/usr/lib/acct/wtmpfix [files]
```

DESCRIPTION

Fwtmp

Fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in *wtmp* to formatted ASCII records. The ASCII version is useful to enable editing, via *ed*(1), bad records or general purpose maintenance of the file.

The argument *-ic* is used to denote that input is in ASCII form, and output is to be written in binary form.

Wtmpfix

Wtmpfix examines the standard input or named files in *wtmp* format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A *-* can be used in place of *files* to indicate the standard input. If time/date corrections are not performed, *acctconl* will fault when it encounters certain date-change records.

Each time the date is set, a pair of date change records are written to */etc/wtmp*. The first record is the old date denoted by the string *old time* placed in the line field and the flag *OLD_TIME* placed in the type field of the *<utmp.h>* structure. The second record specifies the new date and is denoted by the string *new time* placed in the line field and the flag *NEW_TIME* placed in the type field. *Wtmpfix* uses these records to synchronize all time stamps in the file.

In addition to correcting time/date stamps, *wtmpfix* will check the validity of the name field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to *INVALID* and write a diagnostic to the standard error. In this way, *wtmpfix* reduces the chance that *acctconl* will fail when processing connect accounting records.

FILES

```
/etc/wtmp
/usr/include/utmp.h
```

SEE ALSO

acct(1M), *acctcms*(1M), *acctcom*(1), *acctcon*(1M), *acctmerg*(1M), *acctprc*(1M), *acctsh*(1M), *runacct*(1M), *ed*(1), *acct*(2), *acct*(4), *utmp*(4).

GET(1)

NAME

get - get a version of an SCCS file

SYNOPSIS

get [-rSID] [-ccutoff] [-i|list] [-x|list] [-wstring] [-aseq-no.] [-k] [-e] [-l|p] [-p] [-m] [-n] [-s] [-b] [-g] [-t] file ...

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with -. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading s.; (see also *FILES*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

-rSID The SCCS *I*dentification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved [as well as the SID of the version to be eventually created by *delta*(1) if the -e keyletter is also used], as a function of the SID specified.

-ccutoff *Cutoff* date-time, in the form:

YY[MM[DD[HH[MMISS]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, -c7502 is equivalent to -c750228235959. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: "-c77/2/2 9:22:25". Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

!get "-c%E% %U%" s.file

-e Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The -e keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *gets* from editing on the same SID until *delta* is executed or the j (joint edit) flag is set in the SCCS file [see *admin*(1)]. Concurrent use of *get* -e for different SIDs is always allowed.

If the *g-file* generated by *get* with an -e keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the -k keyletter in place of the -e keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file [see *admin(1)*] are enforced when the `-e` keyletter is used.

- `-b` Used with the `-e` keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the `b` flag is not present in the file [see *admin(1)*] or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.
- `-list` A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

```
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID - SID
```

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.
- `-xlist` A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the `-i` keyletter for the *list* format.
- `-k` Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The `-k` keyletter is implied by the `-e` keyletter.
- `-l[p]` Causes a delta summary to be written into an *l-file*. If `-lp` is used then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.
- `-p` Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the `-s` keyletter is used, in which case it disappears.
- `-s` Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.
- `-m` Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.
- `-n` Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the `-m` and `-n` keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the `-m` keyletter generated format.
- `-g` Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.
- `-t` Used to access the most recently created ("top") delta in a given release (e.g., `-r1`), or release and level (e.g., `-r1.2`).
- `-w string` Substitute *string* for all occurrences of `@(#)get.1` 6.2 when getting the file.

GET(1)

-aseq-no. The delta sequence number of the SCCS file delta (version) to be retrieved [see *sccsfile(4)*]. This keyletter is used by the *comb(1)* command; it is not a generally useful keyletter, and users should not use it. If both the **-r** and **-a** keyletters are specified, the **-a** keyletter is used. Care should be taken when using the **-a** keyletter in conjunction with the **-e** keyletter, as the SID of the delta to be created may not be what one expects. The **-r** keyletter can be used with the **-a** and **-e** keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **-e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **-i** keyletter is used included deltas are listed following the notation "Included"; if the **-x** keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	-b Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	-	R < mR and R does <i>not</i> exist	hR.mL**	hR.mL.(mB+1).1
R	-	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	-	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	-	Branch succ.	R.L.B.S	R.L.(mB+1).1

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

Successor.

- † The **-b** keyletter is effective only if the **b** flag [see *admin*(1)] is present in the file. An entry of **-** means "irrelevant".
- * This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

<i>Keyword</i>	<i>Value</i>
%M%	Module name: either the value of the m flag in the file [see <i>admin</i> (1)], or if absent, the name of the SCCS file with the leading s . removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the t flag in the SCCS file [see <i>admin</i> (1)].
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the q flag in the file [see <i>admin</i> (1)].
%C%	Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is <i>not</i> intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string @(#) recognizable by <i>what</i> (1).
%W%	A shorthand notation for constructing <i>what</i> (1) strings for UNIX system program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing <i>what</i> (1) strings for non-UNIX system program files. %A% = %Z%%Y% %M% %I%%Z%

FILES

Several auxiliary files may be created by *get*. These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form *s.module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s**. prefix. For example, *s.xyz.c*, the auxiliary file names would be *xyz.c*, *l.xyz.c*, *p.xyz.c*, and *z.xyz.c*, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **-p** keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **-k** keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

GET(1)

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the `-l` keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

- a. A blank character if the delta was applied;
 - otherwise.
- b. A blank character if the delta was applied or was not applied and ignored;
 - if the delta was not applied and was not ignored.
- c. A code indicating a "special" reason why the delta was or was not applied:
 - "I": Included.
 - "X": Excluded.
 - "C": Cut off (by a `-c` keyletter).
- d. Blank.
- e. SCCS identification (SID).
- f. Tab character.
- g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
- h. Blank.
- i. Login name of person who created *delta*.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an `-e` keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an `-e` keyletter for the same SID until *delta* is executed or the joint edit flag, `j`, [see *admin(1)*] is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the `-i` keyletter argument if it was present, followed by a blank and the `-x` keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

SEE ALSO

admin(1), *delta(1)*, *help(1)*, *prs(1)*, *what(1)*,
scsfile(4) and "Source Code Control System" in the Software Development System manual.

DIAGNOSTICS

Use *help(1)* for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the `-e` keyletter is used.

GETOPT(1)

NAME

getopt — parse command options

SYNOPSIS

set -- getopt optstring \$*

DESCRIPTION

Getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *Optstring* is a string of recognized option letters [see *getopt(3C)*]; if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option -- is used to delimit the end of the options. If it is used explicitly, *getopt* will recognize it; otherwise, *getopt* will generate it; in either case, *getopt* will place it at the end of the options. The positional parameters (\$1 \$2 ...) of the shell are reset so that each option is preceded by a - and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options a or b, as well as the option o, which requires an argument:

```
set -- getopt abo: $*
if [ $? != 0 ]
then
    echo $USAGE
    exit 2
fi
for i in $*
do
    case $i in
    -a | -b)    FLAG=$i; shift;;
    -o)        OARG=$2; shift 2;;
    --)        shift; break;;
    esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO

sh(1), getopt(3C).

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NAME

getty — set terminal type, modes, speed, and line discipline

SYNOPSIS

```
/etc/getty [ -b ] [ -t timeout ] line [ speed [ type [ linedisc ] ] ]
/etc/getty -c file
```

DESCRIPTION

Getty is a program that is invoked by *init*(1M). It is the second process in the series, (*init-getty-login-shell*) that ultimately connects a user with the UNIX system. Initially *getty* generates a system identification message from the values returned by the *uname*(2) system call. Then, if */etc/issue* exists, it outputs this to the user's terminal, followed finally by the login message field for the entry it is using from */etc/gettydefs*. *Getty* reads the user's login name and invokes the *login*(1) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used.

Line is the name of a tty line in */dev* to which *getty* is to attach itself. *Getty* uses this string as the name of a file in the */dev* directory to open for reading and writing. Unless *getty* is invoked with the *-b* flag, *getty* will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The *-t* flag plus *timeout* in seconds, specifies that *getty* should exit if the open on the line succeeds and no one types anything in the specified number of seconds. The optional second argument, *speed*, is a label to a speed and tty definition in the file */etc/gettydefs*. This definition tells *getty* at what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by typing a *<break>* character). The default *speed* is 300 baud. The optional third argument, *type*, is a character string describing to *getty* what type of terminal is connected to the line in question. *Getty* understands the following types:

none	default
vt61	DEC vt61
vt100	DEC vt100
hp45	Hewlett-Packard HP45
c100	Concept 100

The default terminal is *none*; i.e., any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition. The optional fourth argument, *linedisc*, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, LDISC0.

When given no optional arguments, *getty* sets the *speed* of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, new-line characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in */etc/gettydefs*.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately [see *ioctl*(2)].

GETTY(1M)

The user's name is scanned to see if it contains any lowercase alphabetic characters; if not, and if the name is non-empty, the system is told to map any future uppercase characters into the corresponding lowercase characters.

In addition to the standard UNIX system erase and kill characters (**#** and **@**), *getty* also understands **\b** and **^U**. If the user uses a **\b** as an erase, or **^U** as a kill character, *getty* sets the standard erase character and/or kill character to match.

Getty also understands the "standard" ESS2 protocols for erasing, killing and aborting a line, and terminating a line. If *getty* sees the ESS erase character, **_** or kill character, **\$**, or abort character, **&**, or the ESS line terminators, **/** or **!**, it arranges for this set of characters to be used for these functions.

Finally, *login* is called with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which will place them in the environment [see *login*(1)].

A check option is provided. When *getty* is invoked with the **-c** option and *file*, it scans the file as if it were scanning */etc/gettydefs* and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *ioctl*(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

ct(1C), *init*(1M), *login*(1), *ioctl*(2), *gettydefs*(4), *inittab*(4), *tty*(7).

BUGS

While *getty* does understand simple single character quoting conventions, it is not possible to quote the special control characters that *getty* uses to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. Therefore it is not possible to login via *getty* and type a **#**, **@**, **/**, **!**, **_**, backspace, **^U**, **^D**, or **&** as part of your login name or arguments. They will always be interpreted as having their special meaning as described above.

NAME

greek — select terminal filter

SYNOPSIS

greek [-Tterminal]

DESCRIPTION

Greek is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character TELETYPE® Model 37 terminal [which is the *nroff*(1) default terminal] for certain other terminals. Special characters are simulated by overstriking, if necessary and possible. If the argument is omitted, *greek* attempts to use the environment variable *\$TERM* [see *environ*(5)]. The following *terminals* are recognized currently:

300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	DIABLO 1620 (alias DASI 450).
1620-12	DIABLO 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	TEKTRONIX 4014.
hp	Hewlett-Packard 2621, 2640, and 2645.
tek	TEKTRONIX 4014.

FILES

/usr/bin/300
 /usr/bin/300s
 /usr/bin/4014
 /usr/bin/450
 /usr/bin/hp

SEE ALSO

300(1), 4014(1), 450(1), hp(1), tplot(1G),
 environ(5), greek(5), term(5) in the Software Development System manual.

"Mathematics Typesetting Program" (egn), "Memorandum Macros User Guide" and
 "Nroff and Troff User manual" in the Text Preparation System manual.

GREP(1)

NAME

grep, egrep, fgrep — search a file for a pattern

SYNOPSIS

```
grep [ options ] expression [ files ]
egrep [ options ] [ expression ] [ files ]
fgrep [ options ] [ strings ] [ files ]
```

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *grep* patterns are limited regular *expressions* in the style of *ed*(1); it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

- v All lines but those matching are printed.
- x (Exact) only lines matched in their entirety are printed (*fgrep* only).
- c Only a count of matching lines is printed.
- i Ignore upper/lowercase distinction during comparisons.
- l Only the names of files with matching lines are listed (once), separated by new-lines.
- n Each line is preceded by its relative line number in the file.
- b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.
- s The error messages produced for nonexistent or unreadable files are suppressed (*grep* only).
- e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a - (does not work with *grep*).
- f *file*
The regular *expression* (*egrep*) or *strings* list (*fgrep*) is taken from the *file*.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters \$, *, [, ^, |, (,), and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'

Fgrep searches for lines that contain one of the *strings* separated by new-lines.

Egrep accepts regular expressions as in *ed*(1), except for \ (and \), with the addition of:

1. A regular expression followed by + matches one or more occurrences of the regular expression.
2. A regular expression followed by ? matches 0 or 1 occurrences of the regular expression.
3. Two regular expressions separated by | or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses () for grouping.

The order of precedence of operators is [], then *?+, then concatenation, then | and new-line.

SEE ALSO

ed(1), *sed*(1), *sh*(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Ideally there should be only one *grep*, but we do not know a single algorithm that spans a wide enough range of space-time tradeoffs.

Lines are limited to `BUFSIZ` characters; longer lines are truncated. (`BUFSIZ` is defined in `/usr/include/stdio.h`.)

Egrep does not recognize ranges, such as `[a-z]`, in character classes.

If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

HELP(1)

NAME

help — ask for help

SYNOPSIS

help [args]

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

- | | |
|--------|--|
| type 1 | Begins with non-numeric, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., <i>ge6</i> , for message 6 from the <i>get</i> command). |
| type 2 | Does not contain numerics (as a command, such as <i>get</i>) |
| type 3 | Is all numeric (e.g., <i>212</i>) |

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

FILES

<i>/usr/lib/help</i>	directory containing files of message text.
<i>/usr/lib/help/helploc</i>	file containing locations of help files not in <i>/usr/lib/help</i> .

DIAGNOSTICS

Use *help(1)* for explanations.

NAME

hp — handle special functions of HP 2640 and 2621-series terminals

SYNOPSIS

hp [*-e*] [*-m*]

DESCRIPTION

Hp supports special functions of the Hewlett-Packard 2640 series of terminals, with the primary purpose of producing accurate representations of most *nroff* output. A typical use is:

```
nroff -h files ... | hp
```

Regardless of the hardware options on your terminal, *hp* tries to do sensible things with underlining and reverse line-feeds. If the terminal has the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it has the "mathematical-symbol" feature, Greek and other special characters can be displayed.

The flags are as follows:

- e** It is assumed that your terminal has the "display enhancements" feature, and so maximal use is made of the added display modes. Overstruck characters are presented in the Underline mode. Superscripts are shown in Half-bright mode, and subscripts in Half-bright, Underlined mode. If this flag is omitted, *hp* assumes that your terminal lacks the "display enhancements" feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in Inverse Video mode, i.e., dark-on-light, rather than the usual light-on-dark.
- m** Requests minimization of output by removal of new-lines. Any contiguous sequence of 3 or more new-lines is converted into a sequence of only 2 new-lines; i.e., any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other special characters, *hp* provides the same set as does *300(1)*, except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown. The display is adequate for examining output from *neqn*.

DIAGNOSTICS

"line too long" if the representation of a line exceeds 1,024 characters.

The exit codes are 0 for normal termination, 2 for all errors.

SEE ALSO

300(1), *col(1)*, *greek(1)*.

"Nroff and Troff User manual", "Mathematics Typesetting Program" (*eqn*), and "Table Formating Program" (*tbl*) in the Text Preparation System manual.

BUGS

An "overstriking sequence" is defined as a printing character followed by a backspace followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in Inverse Video; otherwise, only the first printing character is shown (again, underlined or in Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (e.g., reverse line-feeds, backspaces) can make text "disappear"; in particular, tables generated by *tbl(1)* that contain vertical lines will often be missing the lines of text that contain the "foot" of a vertical line, unless the input to *hp* is piped through *col(1)*.

Although some terminals do provide numerical superscript characters, no attempt is made to display them.

HPIO(1)

NAME

hpio - HP 2645A terminal tape file archiver

SYNOPSIS

hpio -o[rcl] file ...

hpio -i[rta] [-n count]

DESCRIPTION

Hpio is designed to take advantage of the tape drives on Hewlett-Packard 2645A terminals. Up to 255 UNIX system files can be archived onto a tape cartridge for off-line storage or for transfer to another UNIX system. The actual number of files depends on the sizes of the files. One file of about 115,000 bytes will almost fill a tape cartridge. Almost 300 1-byte files will fit on a tape, but the terminal will not be able to retrieve files after the first 255. This manual page is not intended to be a guide for using tapes on HP 2645A terminals, but tries to give enough information to be able to create and read tape archives and to position a tape for access to a desired file in an archive.

Hpio -o (copy out) copies the specified *file(s)*, together with path name and status information to a tape drive on your terminal (which is assumed to be positioned at the beginning of a tape or immediately after a tape mark). The left tape drive is used by default. Each *file* is written to a separate tape file and terminated with a tape mark. When *hpio* finishes, the tape is positioned following the last tape mark written.

Hpio -i (copy in) extracts a file(s) from a tape drive (which is assumed to be positioned at the beginning of a file that was previously written by a **hpio -o**). The default action extracts the next file from the left tape drive.

Hpio always leaves the tape positioned after the last file read from or written to the tape. Tapes should always be rewound before the terminal is turned off. To rewind a tape depress the green function button, then function key 5, and then select the appropriate tape drive by depressing either function key 5 for the left tape drive or function key 6 for the right. If several files have been archived onto a tape, the tape may be positioned at the beginning of a specific file by depressing the green function button, then function key 8, followed by typing the desired file number (1-255) with no RETURN, and finally function key 5 for the left tape or function key 6 for the right. The desired file number may also be specified by a signed number relative to the current file number.

The meanings of the available options are:

- r** Use the right tape drive.
- c** Include a checksum at the end of each *file*. The checksum is always checked by **hpio -i** for each file written with this option by **hpio -o**.
- n count** The number of input files to be extracted is set to *count*. If this option is not given, *count* defaults to 1. An arbitrarily large *count* may be specified to extract all files from the tape. *Hpio* will stop at the end of data mark on the tape.
- t** Print a table of contents only. No files are created. Printed information gives the file size in bytes, the file name, the file access modes, and whether or not a checksum is included for the file.
- a** Ask before creating a file. **Hpio -i** normally prints the file size and name, creates and reads in the file, and prints a status message when the file has been read in. If a checksum is included with the file, it reports whether the checksum matched its computed value. With this option, the file size and name are printed followed by a ?. Any response beginning with **y** or **Y** will cause the file to be copied in as above. Any other response will cause the file to be skipped.

FILES

/dev/tty?? to block messages while accessing a tape

SEE ALSO

cu(1C).

DIAGNOSTICS

BREAK

An interrupt signal terminated processing.

Can't create '*file*'.

File system access permissions did not allow *file* to be created.

Can't get tty options on stdout.

Hpio was unable to get the input-output control settings associated with the terminal.

Can't open '*file*'.

File could not be accessed to copy it to tape.

End of Tape.

No tape record was available when a read from a tape was requested. An end of data mark is the usual reason for this, but it may also occur if the wrong tape drive is being accessed and no tape is present.

'*file*' not a regular file.

File is a directory or other special file. Only regular files will be copied to tape.

Readent = *rc*, termcnt = *tc*.

Hpio expected to read *rc* bytes from the next block on the tape, but the block contained *tc* bytes. This is caused by having the tape improperly positioned or by a tape block being mangled by interference from other terminal I/O.

Skip to next file failed.

An attempt to skip over a tape mark failed.

Tape mark write failed.

An attempt to write a tape mark at the end of a file failed.

Write failed.

A tape write failed. This is most frequently caused by specifying the wrong tape drive, running off the end of the tape, or trying to write on a tape that is write protected.

WARNINGS

Tape I/O operations may copy bad data if any other I/O involving the terminal occurs. Do not attempt any type ahead while *hpio* is running. *Hpio* turns off write permissions for other users while it is running, but processes started asynchronously from your terminal can still interfere. The most common indication of this problem, while a tape is being written, is the appearance of characters on the display screen that should have been copied to tape.

The keyboard, including the terminal BREAK key, is locked during tape write operations; the BREAK key is only functional between writes.

Hpio must have complete control of the attributes of the terminal to communicate with the tape drives. Interaction with commands such as *cu*(1C) may interfere and prevent successful operation.

BUGS

Some binary files contain sequences that will confuse the terminal.

An *hpio -i* that encounters the end of data mark on the tape (e.g., scanning the entire tape with *hpio -itn 300*), leaves the tape positioned *after* the end of data mark. If a subsequent *hpio -o* is done at this point, the data will not be retrievable. The tape must be repositioned manually using the terminal FIND FILE -1 operation (depress the green function button, function key 8, and then function key 5 for the left tape or function key 6 for the right tape) before the

HPIO(1)

hpio -o is started.

If an interrupt is received by *hpio* while a tape is being written, the terminal may be left with the keyboard locked. If this happens, the terminal's RESET TERMINAL key will unlock the keyboard.

NAME hyphen -- find hyphenated words

SYNOPSIS
hyphen [files]

DESCRIPTION
Hyphen finds all the hyphenated words ending lines in *files* and prints them on the standard output. If no arguments are given, the standard input is used; thus, *hyphen* may be used as a filter.

EXAMPLE
The following will allow the proofreading of *nroff* hyphenation in *textfile*.
mm textfile | hyphen

SEE ALSO
"Memorandum Macros User Guide" and "Nroff and Troff Usermanual" in the Text Preparation System manual.

BUGS
Hyphen cannot cope with hyphenated *italic* (i.e., underlined) words; it will often miss them completely, or mangle them.
Hyphen occasionally gets confused, but with no ill effects other than spurious extra output.

IB(1M)

NAME

`ib` - install boot image

SYNOPSIS

`ib device1 device2 [bootfile]`

DESCRIPTION

ib creates a bootstrap file from *bootfile* and installs it on the specified device(s). If the *bootfile* is not specified, *ib* creates a bootstrap file from */etc/stlboot*.

If the bootstrap file is installed on a floppy disk, then *device1* should specify the single-sided, single density, 128-byte sectored first track; followed by *device2*, which should specify the next track of the floppy disk. If the bootstrap file is installed on a hard disk, then *device1* should specify the 1024-byte sectored first track; followed by *device2*, which should specify the device on which the root file system is to be placed by *mkfs(1M)*.

For example, to install the bootstrap file, *letchmyboot*, onto a floppy you can enter:

```
ib /dev/rdisk/0s15 /dev/dsk/0s24 /etc/myboot
```

the file is installed split across the end of the first 512 bytes, which are used to contain a media label.

FILES

<i>/etc/stlboot</i>	the normal bootstrap file
<i>/dev/rdisk/*</i>	the raw disk devices

SEE ALSO

stlboot(1M).

NAME

id — print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

Id writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

SEE ALSO

logname(1).

getuid(2) in the Software Development System manual.

INIT(1M)

NAME

init, telinit — process control initialization

SYNOPSIS

`/etc/init [0123456SsQq]`

`/etc/telinit [0123456sSQqabc]`

DESCRIPTION

Init

Init is a general process spawner. Its primary role is to create processes from a script stored in the file `/etc/inittab` [see *inittab(4)*]. This file usually has *init* spawn *getty*'s on each line that a user may log in on. It also controls autonomous processes required by any particular system.

Init considers the system to be in a *run-level* at any given time. A *run-level* can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by *init* for each of these *run-levels* is defined in the *inittab* file. *Init* can be in one of eight *run-levels*, 0–6 and S or s. The *run-level* is changed by having a privileged user run `/etc/init` (which is linked to *telinit*). This user-spawned *init* sends appropriate signals to the original *init* spawned by the operating system when the system was rebooted, telling it which *run-level* to change to.

Init is invoked inside the UNIX system as the last step in the boot procedure. The first thing *init* does is to look for `/etc/inittab` and see if there is an entry of the type *initdefault* [see *inittab(4)*]. If there is, *init* uses the *run-level* specified in that entry as the initial *run-level* to enter. If this entry is not in *inittab* or *inittab* is not found, *init* requests that the user enter a *run-level* from the virtual system console, `/dev/syscon`. If an S (s) is entered, *init* goes into the *SINGLE USER* level. This is the only *run-level* that doesn't require the existence of a properly formatted *inittab* file. If `/etc/inittab` doesn't exist, then by default the only legal *run-level* that *init* can enter is the *SINGLE USER* level. In the *SINGLE USER* level the virtual console terminal `/dev/syscon` is opened for reading and writing and the command `/bin/su` is invoked immediately. To exit from the *SINGLE USER run-level* one of two options can be elected. First, if the shell is terminated (via an end-of-file), *init* will reprompt for a new *run-level*. Second, the *init* or *telinit* command can signal *init* and force it to change the *run-level* of the system.

When attempting to boot the system, failure of *init* to prompt for a new *run-level* may be due to the fact that the device `/dev/syscon` is linked to a device other than the physical system teletype (`/dev/systty`). If this occurs, *init* can be forced to relink `/dev/syscon` by typing a delete on the system teletype which is collocated with the processor.

When *init* prompts for the new *run-level*, the operator may enter only one of the digits 0 through 6 or the letters S or s. If S is entered *init* operates as previously described in *SINGLE USER* mode with the additional result that `/dev/syscon` is linked to the user's terminal line, thus making it the virtual system console. A message is generated on the physical console, `/dev/systty`, saying where the virtual terminal has been relocated.

When *init* comes up initially and whenever it switches out of *SINGLE USER* state to normal run states, it sets the *ioctl(2)* states of the virtual console, `/dev/syscon`, to those modes saved in the file `/etc/ioctl.syscon`. This file is written by *init* whenever *SINGLE USER* mode is entered. If this file does not exist when *init* wants to read it, a warning is printed and default settings are assumed.

If a 0 through 6 is entered *init* enters the corresponding *run-level*. Any other input will be rejected and the user will be re-prompted. If this is the first time *init* has entered a *run-level* other than *SINGLE USER*, *init* first scans *inittab* for special entries of the type *boot* and *bootwait*. These entries are performed, providing the *run-level* entered matches that of the entry before any normal processing of *inittab* takes place. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. The *inittab* file is scanned to find all entries that are to be processed for that *run-level*.

Run-level 2 is usually defined by the user to contain all of the terminal processes and daemons that are spawned in the multiuser environment.

In a multiuser environment, the *inittab* file is usually set up so that *init* will create a process for each terminal on the system.

For terminal processes, ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as the result of hanging up. When *init* receives a child death signal, telling it that a process it spawned has died, it records the fact and the reason it died in */etc/utmp* and */etc/wtmp* if it exists [see *who(1)*]. A history of the processes spawned is kept in */etc/wtmp* if such a file exists.

To spawn each process in the *inittab* file, *init* reads each entry and for each entry which should be respawned, it forks a child process. After it has spawned all of the processes specified by the *inittab* file, *init* waits for one of its descendant processes to die, a powerfail signal, or until *init* is signaled by *init* or *telinit* to change the system's *run-level*. When one of the above three conditions occurs, *init* re-examines the *inittab* file. New entries can be added to the *inittab* file at any time; however, *init* still waits for one of the above three conditions to occur. To provide for an instantaneous response the *init Q* or *init q* command can wake *init* to re-examine the *inittab* file.

If *init* receives a *powerfail* signal (*SIGPWR*) and is not in *SINGLE USER* mode, it scans *inittab* for special powerfail entries. These entries are invoked (if the *run-levels* permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions whenever the operating system experiences a power failure.

When *init* is requested to change *run-levels* (via *telinit*), *init* sends the warning signal (*SIGTERM*) to all processes that are undefined in the target *run-level*. *Init* waits 20 seconds before forcibly terminating these processes via the kill signal (*SIGKILL*).

Telinit

Telinit, which is linked to *lctcknit*, is used to direct the actions of *init*. It takes a one-character argument and signals *init* via the kill system call to perform the appropriate action. The following arguments serve as directives to *init*.

- 0-6** tells *init* to place the system in one of the *run-levels* 0-6.
- a,b,c** tells *init* to process only those */etc/inittab* file entries having the **a**, **b** or **c** *run-level* set.
- Q,q** tells *init* to re-examine the */etc/inittab* file.
- s,S** tells *init* to enter the single user environment. When this level change is effected, the virtual system teletype, */dev/syscon*, is changed to the terminal from which the command was executed.

Telinit can only be run by someone who is super-user or a member of group *sys*.

INIT(1M)

FILES

/etc/inittab
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/dev/syscon
/dev/systty

SEE ALSO

getty(1M), *login(1)*, *sh(1)*, *who(1)*, *kill(2)*, *inittab(4)*, *utmp(4)*.

DIAGNOSTICS

If *init* finds that it is continuously respawning an entry from */etc/inittab* more than 10 times in 2 minutes, it will assume that there is an error in the command string, and generate an error message on the system console, and refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in the *inittab*.

NAME

inittab — script for the *init* process

DESCRIPTION

The *inittab* file supplies the script to *init*'s role as a general process dispatcher. The process that constitutes the majority of *init*'s process dispatching activities is the line process */etc/getty* that initiates individual terminal lines. Other processes typically dispatched by *init* are daemons and the shell.

The *inittab* file is composed of entries that are position dependent and have the following format:

```
id:rstate:action:process
```

Each entry is delimited by a new-line; however, a backslash (\) preceding a new-line indicates a continuation of the entry. Up to 512 characters per entry are permitted. Comments may be inserted in the *process* field using the *sh*(1) convention for comments. Comments for lines that spawn *gettys* are displayed by the *who*(1) command. It is expected that they will contain some information about the line such as the location. There are no limits (other than maximum entry size) imposed on the number of entries within the *inittab* file. The entry fields are:

id This is one or two characters used to uniquely identify an entry.

rstate This defines the *run-level* in which this entry is to be processed. *Run-levels* effectively correspond to a configuration of processes in the system. That is, each process spawned by *init* is assigned a *run-level* or *run-levels* in which it is allowed to exist. The *run-levels* are represented by a number ranging from 0 through 6. As an example, if the system is in *run-level* 1, only those entries having a 1 in the *rstate* field will be processed. When *init* is requested to change *run-levels*, all processes which do not have an entry in the *rstate* field for the target *run-level* will be sent the warning signal (SIGTERM) and allowed a 20-second grace period before being forcibly terminated by a kill signal (SIGKILL). The *rstate* field can define multiple *run-levels* for a process by selecting more than one *run-level* in any combination from 0-6. If no *run-level* is specified, then the process is assumed to be valid at all *run-levels* 0-6. There are three other values, a, b and c, which can appear in the *rstate* field, even though they are not true *run-levels*. Entries which have these characters in the *rstate* field are processed only when the *telinit* [see *init*(1M)] process requests them to be run (regardless of the current *run-level* of the system). They differ from *run-levels* in that *init* can never enter *run-level* a, b or c. Also, a request for the execution of any of these processes does not change the current *run-level*. Furthermore, a process started by an a, b or c command is not killed when *init* changes levels. They are only killed if their line in */etc/inittab* is marked off in the *action* field, their line is deleted entirely from */etc/inittab*, or *init* goes into the *SINGLE USER* state.

action Key words in this field tell *init* how to treat the process specified in the *process* field. The actions recognized by *init* are as follows:

respawn If the process does not exist then start the process, do not wait for its termination (continue scanning the *inittab* file), and when it dies restart the process. If the process currently exists then do nothing and continue scanning the *inittab* file.

wait Upon *init*'s entering the *run-level* that matches the entry's *rstate*, start the process and wait for its

- termination. All subsequent reads of the *inittab* file while *init* is in the same *run-level* will cause *init* to ignore this entry.
- once** Upon *init*'s entering a *run-level* that matches the entry's *rstate*, start the process, do not wait for its termination. When it dies, do not restart the process. If upon entering a new *run-level*, where the process is still running from a previous *run-level* change, the program will not be restarted.
- boot** The entry is to be processed only at *init*'s boot-time read of the *inittab* file. *Init* is to start the process, not wait for its termination; and when it dies, not restart the process. In order for this instruction to be meaningful, the *rstate* should be the default or it must match *init*'s *run-level* at boot time. This action is useful for an initialization function following a hardware reboot of the system.
- bootwait** The entry is to be processed only at *init*'s boot-time read of the *inittab* file. *Init* is to start the process, wait for its termination and, when it dies, not restart the process.
- powerfail** Execute the process associated with this entry only when *init* receives a power fail signal [SIGPWR see *signal*(2)].
- powerwait** Execute the process associated with this entry only when *init* receives a power fail signal (SIGPWR) and wait until it terminates before continuing any processing of *inittab*.
- off** If the process associated with this entry is currently running, send the warning signal (SIGTERM) and wait 20 seconds before forcibly terminating the process via the kill signal (SIGKILL). If the process is nonexistent, ignore the entry.
- ondemand** This instruction is really a synonym for the **respawn** action. It is functionally identical to **respawn** but is given a different keyword in order to divorce its association with *run-levels*. This is used only with the *a*, *b* or *c* values described in the *rstate* field.
- initdefault** An entry with this *action* is only scanned when *init* initially invoked. *Init* uses this entry, if it exists, to determine which *run-level* to enter initially. It does this by taking the highest *run-level* specified in the *rstate* field and using that as its initial state. If the *rstate* field is empty, this is interpreted as **0123456** and so *init* will enter *run-level* 6. Also, the **initdefault** entry cannot specify that *init* start in the *SINGLE USER* state. Additionally, if *init* does not find an **initdefault** entry in */etc/inittab*, then it will request an initial *run-level* from the user at reboot time.
- sysinit** Entries of this type are executed before *init* tries to access the console. It is expected that this entry will be only used to initialize devices on which *init* might try to ask the *run-level* question. These entries are executed and waited for before continuing.

process This is a *sh* command to be executed. The entire *process* field is prefixed with *exec* and passed to a forked *sh* as *sh -c 'exec command'*. For this reason, any legal *sh* syntax can appear in the *process* field. Comments can be inserted with the *;*comment** syntax.

FILES

/etc/inittab

SEE ALSO

exec(2), *open(2)*, *signal(2)* in the Software Development System manual.
gety(1M), *init(1M)* *sh(1)*, *who(1)*.

INSTALL(1M)

NAME

install — install commands

SYNOPSIS

/etc/install [-c *dira*] [-f *dirb*] [-i] [-n *dirc*] [-o] [-s] *file* [*dirx* ...]

DESCRIPTION

Install is a command most commonly used in "makefiles" [see *make(1)*] to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (*dirx* ...) are given, *install* will search a set of default directories (*/bin*, */usr/bin*, */etc*, */lib*, and */usr/lib*, in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx* ...) are specified after *file*, those directories will be searched before the directories specified in the default list.

The meanings of the options are:

- c *dira* Installs a new command (*file*) in the directory specified by *dira*, only if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the -s option.
- f *dirb* Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to 755 and *bin*, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the -o or -s options.
- i Ignores default directory list, searching only through the given directories (*dirx* ...). May be used alone or with any other options other than -c and -f.
- n *dirc* If *file* is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file will be set to 755 and *bin*, respectively. May be used alone or with any other options other than -c and -f.
- o If *file* is found, this option saves the "found" file by copying it to *OLDfile* in the directory in which it was found. This option is useful when installing a normally text busy file such as */bin/sh* or */etc/getty*, where the existing file cannot be removed. May be used alone or with any other options other than -c.
- s Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

cpset(1M), *make(1)*, *mk(8)*.

NAME

installit—package installation

SYNOPSIS

installit

DESCRIPTION

Installit performs package installation. Certain package installations may require the presence of other programs and or utilities. Check package descriptions under the Installation Information.

FILES

The following files are created during the installation process;

/makelink
/execfile
/diskdata

SEE ALSO

divvy(1M), fdisk(1M), showbad(1M).
"Installation Instructions."

IPCRM(1)

NAME

ipcrm - remove a message queue, semaphore set or shared memory id

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

Ipcrm will remove one or more specified messages, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

- q *msqid*** removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.
- m *shmid*** removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
- s *semid*** removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.
- Q *msgkey*** removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.
- M *shmkey*** removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
- S *semkey*** removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in *msgctl(2)*, *shmctl(2)*, and *semctl(2)*. The identifiers and keys may be found by using *ipcs(1)*.

SEE ALSO

ipcs(1),
msgctl(2), *msgget(2)*, *msgop(2)*, *semctl(2)*, *semget(2)*, *semop(2)*, *shmctl(2)*, *shmget(2)*,
shmop(2) in the Software Development System manual.

NAME

ipcs — report interprocess communication facilities status

SYNOPSIS

ipcs [options]

DESCRIPTION

Ipcs prints certain information about active interprocess communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following *options*:

380.sp40u

- q Print information about active message queues.
- m Print information about active shared memory segments.
- s Print information about active semaphores.

If any of the options *-q*, *-m*, or *-s* are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed.

- b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.
- c Print creator's login name and group name. See below.
- o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)
- p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.
- t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, last *semop*(2) on semaphores.) See below.
- a Use all print *options*. (This is a shorthand notation for *-b*, *-c*, *-o*, *-p*, and *-t*.)
- C *corefile*
Use the file *corefile* in place of */dev/kmem*.
- N *namelist*
The argument will be taken as the name of an alternate *namelist* (*/unix* is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each

IPCS(1)

facility; they do *not* determine which facilities will be listed.

T	(all)	Type of the facility: <ul style="list-style-type: none">q message queue;m shared memory segment;s semaphore.
ID	(all)	The identifier for the facility entry.
KEY	(all)	The key used as an argument to <i>msgget</i> , <i>semget</i> , or <i>shmget</i> to create the facility entry. (Note: The key of a shared memory segment is changed to <code>IPC_PRIVATE</code> when the segment has been removed until all processes attached to the segment detach it.)
MODE	(all)	The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows: The first two characters are: <ul style="list-style-type: none">R if a process is waiting on a <i>msgrcv</i>;S if a process is waiting on a <i>msgsnd</i>;D if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;C if the associated shared memory segment is to be cleared when the first attach is executed;- if the corresponding special flag is not set. The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused. The permissions are indicated as follows: <ul style="list-style-type: none">r if read permission is granted;w if write permission is granted;a if alter permission is granted;- if the indicated permission is <i>not</i> granted.
OWNER	(all)	The login name of the owner of the facility entry.
GROUP	(all)	The group name of the group of the owner of the facility entry.
CREATOR	(a,c)	The login name of the creator of the facility entry.
CGROUP	(a,c)	The group name of the group of the creator of the facility entry.
CBYTES	(a,o)	The number of bytes in messages currently outstanding on the associated message queue.
QNUM	(a,o)	The number of messages currently outstanding on the associated message queue.

QBYTES	(a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue.
ISPID	(a,p)	The process ID of the last process to send a message to the associated queue.
LRPID	(a,p)	The process ID of the last process to receive a message from the associated queue.
STIME	(a,t)	The time the last message was sent to the associated queue.
RTIME	(a,t)	The time the last message was received from the associated queue.
CTIME	(a,t)	The time when the associated entry was created or changed.
NATTCH	(a,o)	The number of processes attached to the associated shared memory segment.
SEGSZ	(a,b)	The size of the associated shared memory segment.
CPID	(a,p)	The process ID of the creator of the shared memory entry.
LPID	(a,p)	The process ID of the last process to attach or detach the shared memory segment.
ATIME	(a,t)	The time the last attach was completed to the associated shared memory segment.
DTIME	(a,t)	The time the last detach was completed on the associated shared memory segment.
NSEMS	(a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME	(a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/unix	system name list
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

msgop(2), semop(2), shmop(2) in the Software Development System manual.

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

JOIN(1)

NAME

join - relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is -, the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the below options use the argument *n*. This argument should be a 1 or a 2 referring to either *file1* or *file2*, respectively. The following options are recognized:

- an In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
- e *s* Replace empty output fields by string *s*.
- jn *m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with 1.
- o *list* Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.
- tc Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

```
join -jl 4 -j2 3 -o 1.1 2.1 1.6 -t: /etc/passwd /etc/group
```

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

BUGS

With default field separation, the collating sequence is that of `sort -b`; with `-t`, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq* and *awk*(1) are wildly incongruous.

File names that are numeric may cause conflict when the `-o` option is used right before listing file names.

NAME

keyset, also called setkey — programmable function keys

SYNOPSIS

```
setkey [shift-mod] [shift-mod] ... key 'newstr'
setkey -p
setkey -dkey
```

DESCRIPTION

Setkey programs the following special keys on the system console:

center	del	down	end	home	ins	left	minus
pgdn	pgup	plus	prts	right	tab	up	f1
f2	f3	f4	f5	f6	f7	f8	f9
f10	f11	f12	f13	f14	f15	f16	f17
f18	f19	f20	f21	f22	f23	f24	f25
f26	f27	f28	f29	f30			

The key 'center' is the '5' key on the numeric keypad. 'Left', 'right', 'up', and 'down' refer to the arrow keys. The 'shift-mod' argument may be any combination of 'shift', 'alt' and 'control' (or 'ctrl') separated by spaces. The function keys f11 through f20 are the corresponding f1 through f10 keys with a shift modifier of 'shift'. The function keys f21 through f30 are the corresponding f1 through f10 keys with a shift modifier of 'control'.

The 'newstr' argument is the replacement string to be sent when the corresponding key combination is pressed. To insert control characters in the 'newstr' argument, use the '^X' form for control character 'X' (and '^A' for '^'). If the 'newstr' argument is not present, the current value for the key is printed (to stdout). If the 'newstr' is a NULL string (ie., ""), the replacement string is deleted.

The 'setkey -p' command prints the values of all the key settings. It inserts the command name as the first argument so you can redirect the output to a file, edit the file, and execute the file to change the keymap settings.

The 'setkey -d key' command deletes the keymap setting for 'key'. If 'key' is not specified, all the keymap settings are deleted.

'Setkey' by itself prints a help message, with the text of the message printed on stderr and the list of keys printed on stdout so it can be redirected to a file.

There are a number of default settings, corresponding to the ANSI sequences, that are initialized for the system console in the */etc/rc.d/keybrd.rc* file when the system comes up multi-user. In addition, the *setkey* affects only the current virtual console unless it's on the system console, in which case it's also the default for the other consoles. This also implies deleting the keymap on a virtual console (other than the system console) causes the keymap to revert to the defaults established for the system console.

BUGS

There is no way to distinguish the keys f11 through f30 with their shifted counterparts (ie., f11 is the same as shift f1). Therefore, the following (for example) can occur:

```
Setting:                setkey f11 'format /dev/rdisk/0s24^M'
Printing (setkey -p):  setkey shift f1 'format /dev/rdisk/0s24^M'
```

KILL(1)

NAME

kill – terminate a process

SYNOPSIS

kill [-signo] PID ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with **&** is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(1).

The details of the **kill** are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The **killed** process must belong to the current user unless he is the super-user.

If a signal number preceded by **-** is given as first argument, that signal is sent instead of terminate (see *signal*(2)). In particular “kill -9 ...” is a sure **kill**.

SEE ALSO

ps(1), *sh*(1).

kill(2), *signal*(2) in the Software Development System manual.

NAME

killall - kill all active processes

SYNOPSIS

/etc/killall [*signal*]

DESCRIPTION

Killall is a procedure used by */etc/shutdown* to kill all active processes not directly related to the shutdown procedure.

Killall is chiefly used to terminate all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

Killall sends *signal* [see *kill(1)*] to all remaining processes not belonging to the above group of exclusions. If no *signal* is specified, a default of 9 is used.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), *kill(1)*, *ps(1)*, *shutdown(1M)*, *signal(2)*.

LD(1)

NAME

ld - link editor for object files

SYNOPSIS

ld [options] filenames ...

DESCRIPTION

The *ld* command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and *ld* combines them, producing an object module that can either be executed or used as input for a subsequent *ld* run. The output of *ld* is left in *a.out*. This file is executable on the target machine if no errors occurred during the load. If any input file, *filename*, is not an object file, *ld* assumes it is either a text file containing link editor directives or an archive library. (See *The Link Editor User's Manual* for a discussion of input directives.)

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. The order of library members is unimportant because *ld* passes through each library's (archive) symbol table [see *ar(4)*] as many times as necessary until no new external symbols are resolved and no new references are generated. *Ld* will only load together files with the same magic number (large model code cannot be loaded with small model code).

The following options are recognized by *ld*.

- a Produce an absolute executable file; give warnings for undefined references. Relocation information is stripped from the output object file unless the *-r* option is given. The *-r* option is needed only when an absolute file should retain its relocation information (not the normal case). If neither *-a* nor *-r* is given, *-a* is assumed.
- e *epsym* Set the default entry point address for the output file to be that of the symbol *epsym*. This option forces the *-X* option to be set.
- f *fill* Set the default fill pattern for "holes" within an output section as well as initialized bss sections. The argument *fill* is a two-byte constant.
- l*x* Search a library *libx.a*, where *x* is up to seven characters. A library is searched when its name is encountered, so the placement of a *-l* is significant; *-l* options should appear on the command line after any files that reference symbols defined in the library. By default, libraries are located in */lib* and */usr/lib*.
- m Produce a map or listing of the input/output sections on the standard output.
- o *outfile* Produce an output object file by the name *outfile*. The name of the default object file is *a.out*.
- r Retain relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *ld* run. Unless *-a* is also given, the link editor will not complain about unresolved references, and the output file will not be executable.

- s Strip line number entries and symbol table information from the output object file.
- t Turn off the warning about multiply-defined symbols that are not the same size.
- u *symname* Enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. This option must precede the library where the symbol is defined.
- x Do not preserve local (non-globl) symbols in the output symbol table; enter external and static symbols only. This option saves some space in the output file.
- k *nn* For small model programs, specifies the size of the run-time stack. *Nn* is the number of bytes to be allocated. The default is 0X200.
- K For large model programs, specifies that sizes of text, data, and bss should be actual byte counts rather than byte counts rounded up to the nearest "click" size ("click size" as defined in /usr/include/sys/sysmacros.h). This option is provided for operating system support only.
- L *dir* Change the algorithm of searching for *libx.a* to look in *dir* before looking in /lib and /usr/lib. This option is effective only if it precedes the -l option on the command line.
- M Print a warning message for each multiply-defined symbol.
- N Put the data section immediately following the text in the output file.
- V Output a message giving information about the version of *ld* being used.
- VS *num* Use *num* as a decimal version stamp identifying the *a.out* file that is produced. The version stamp is stored in the optional header.

FILES

LIBDIR/libx.a	libraries
a.out	output file

SEE ALSO

as(1), cc(1).
a.out(4), ar(4) in the Software Development System manual.

CAVEATS

Through its options and input directives, the link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives and options should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the data space.
- When the link editor is called through *cc(1)*, a startup routine is linked with the user's program. This routine calls *exit()* [see *exit(2)*] after execution of the main program. If the user calls the link editor directly, then the user must insure that the program

LD(1)

always calls `exit()` rather than falling through the end of the entry routine.

The symbols `etext`, `edata`, and `end` are reserved, and if referred to, are set to the first location above the program text, the first location above initialized data, and the first location above all data respectively. These symbols are defined by the link editor and it is erroneous for a user program to redefine them.

If the link editor does not recognize an input file as an object file or an archive file, it will assume it contains link editor directives and attempt to parse it. This will occasionally produce an error message complaining about "syntax errors".

BUGS

The `-VS num` option has an effect only when the `-X` option is also selected.

NAME

lex — generate programs for simple lexical tasks

SYNOPSIS

lex [-rctvn] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text.

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A file *lex.yy.c* is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in *yytext*, an external character array. Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in `[abx-z]` to indicate a, b, x, y, and z; and the operators *, +, and ? mean respectively any non-negative number of, any positive number of, and either zero or one occurrences of, the previous character or character class. The character . is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported. The notation *r{d,e}* in a rule indicates between *d* and *e* instances of regular expression *r*. It has higher precedence than |, but lower than *, ?, +, and concatenation. The character ^ at the beginning of an expression permits a successful match only immediately after a new-line, and the character \$ at the end of an expression requires a trailing new-line. The character / in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within " symbols or preceded by \. Thus `[a-zA-Z]+` matches a string of letters.

Three subroutines defined as macros are expected: `input()` to read a character; `unput(c)` to replace a character read; and `output(c)` to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named `yylex()`, and the library contains a `main()` which calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function `yymore()` accumulates additional characters into the same *yytext*; and the function `yyless(p)` pushes back the portion of the string matched beginning at *p*, which should be between *yytext* and *yytext+yy leng*. The macros `input` and `output` use files `yyin` and `yyout` to read from and write to, defaulted to `stdin` and `stdout`, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes %% it is copied into the external definition area of the *lex.yy.c* file. All rules should follow a %, as in YACC. Lines preceding % which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with (). Note that curly brackets do not imply parentheses; only string substitution is done.

LINK(1M)

NAME

link, unlink – exercise link and unlink system calls

SYNOPSIS

```
/etc/link path1 path2
/etc/unlink path
```

DESCRIPTION

Path1 names an existing file; *path2* names a new directory entry to be created. *Link* creates a new link (directory entry) for the existing file. *Unlink* removes the named directory entry. When the last directory entry for a file has been removed (and no process has the file open), the file ceases to exist.

SEE ALSO

mm(1).
link(2), unlink(2) in the Software Development System manual.

WARNINGS

Link and *unlink* perform their respective system calls on their arguments, abandoning all error checking. These commands may only be executed by the super-user, who (it is hoped) knows what he or she is doing.

NAME

lint - a C program checker

SYNOPSIS

lint [option] ... file ...

DESCRIPTION

Lint attempts to detect features of the C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

Arguments whose names end with *.c* are taken to be C source files. Arguments whose names end with *.ln* are taken to be the result of an earlier invocation of *lint* with either the *-c* or the *-o* option used. The *.ln* files are analogous to *.o* (object) files that are produced by the *cc(1)* command when given a *.c* file as input. Files with other suffixes are warned about and ignored.

Lint will take all the *.c*, *.ln*, and *lib-lx.ln* (specified by *-lx*) files and process them in their command line order. By default, *lint* appends the standard C lint library (*lib-lc.ln*) to the end of the list of files. However, if the *-p* option is used, the portable C lint library (*lib-port.ln*) is appended instead. When the *-c* option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the *-c* option is used, the *.ln* and the *lib-lx.ln* files are ignored.

Lint now refers to a 'small' model version of the appropriate library, or a 'large' model version, depending on the *-M* command line argument. For *-Ml*, the large model version is chosen. The 'small' model libraries are located in *usr/lib/small*, and the 'large' model libraries in *usr/lib/large*.

Any number of *lint* options may be used, in any order, intermixed with file-name arguments. The following options are used to suppress certain kinds of complaints:

- a Suppress complaints about assignments of long values to variables that are not long.
- b Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in many such complaints).
- h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.
- u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program).
- v Suppress complaints about unused arguments in functions.
- x Do not report variables referred to by external declarations but never used.

LINT(1)

The following arguments alter *lint*'s behavior:

- lx Include additional lint library *llib-lx.ln*. For example, you can include a lint version of the Math Library *llib-lm.ln* by inserting *-lm* on the command line. This argument does not suppress the default use of *llib-lc.ln*. These lint libraries must be in the assumed directory. This option can be used to reference local lint libraries and is useful in the development of multifile projects.
- n Do not check compatibility against either the standard or the portable lint library.
- p Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.
- c Cause *lint* to produce a *.ln* file for every *.c* file on the command line. These *.ln* files are the product of *lint*'s first pass only, and are not checked for interfunction compatibility.
- o lib Cause *lint* to create a lint library with the name *llib-llib.ln*. The *-c* option nullifies any use of the *-o* option. The lint library produced is the input that is given to *lint*'s second pass. The *-o* option simply causes this file to be saved in the named lint library. To produce a *llib-llib.ln* without extraneous messages, use of the *-x* option is suggested. The *-v* option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file *llib-lc* is written). These option settings are also available through the use of "lint comments" (see below).
- Ml Check the program for compatibility using the large memory model. See Programming Procedures for UNIX System V/AT.
- Ms Check the program for compatibility using the small memory model. This model is used by default when no memory model is specified. See Programming Procedures for UNIX System V/AT.

The *-D*, *-U*, and *-I* options of *cpp(1)* and the *-g* and *-O* options of *cc(1)* are also recognized as separate arguments. The *-g* and *-O* options are ignored, but, by recognizing these options, *lint*'s behavior is closer to that of the *cc(1)* command. Other options are warned about and ignored. The preprocessor symbol "lint" is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Certain conventional comments in the C source will change the behavior of *lint*:

*/*NOTREACHED*/*

at appropriate points stops comments about unreachable code. [This comment is typically placed just after calls to functions like *exit(2)*].

*/*VARARGSn*/*

suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

*/*ARGSUSED*/*

turns on the *-v* option for the next function.

/*LINTLIBRARY*/

at the beginning of a file shuts off complaints about unused functions and function arguments in this file. This is equivalent to using the `-v` and `-x` options.

Lint produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the `-c` option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

The behavior of the `-c` and the `-o` options allows for incremental use of *lint* on a set of C source files. Generally, one invokes *lint* once for each source file with the `-c` option. Each of these invocations produces a `.ln` file which corresponds to the `.c` file, and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the `-c` option), listing all the `.ln` files with the needed `-lx` options. This will print all the interfile inconsistencies. This scheme works well with *make*(1); it allows *make* to be used to *lint* only the source files that have been modified since the last time the set of source files were *linted*.

FILES

<code>/usr/lib/small</code>	the directories where the <i>lint</i> libraries specified by the option
<code>usr/lib/large</code>	<code>-lx</code> option must exist
<code>linr[12]</code>	first and second passes
<code>llib-1c.ln</code>	declarations for C Library functions (binary format; source is in <code>/usr/lib/llib-1c</code>)
<code>llib-port.ln</code>	declarations for portable functions (binary format; source is in <code>/usr/lib/llib-port</code>)
<code>llib-1m.ln</code>	declarations for Math Library functions (binary format; source is in <code>/usr/lib/llib-1m</code>)
<code>/usr/tmp/*lint*</code>	temporaries

SEE ALSO

`cc`(1), `cpp`(1), `make`(1).

BUGS

Exit(2) and other functions that do not return are not understood; this causes various lies.

LOGIN(1)

NAME

login — sign on

SYNOPSIS

login [name [env-var ...]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *ctrl-d* to indicate an “end-of-file.” (See *How to Get Started* at the beginning of this volume for instructions on how to dial up initially.)

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

```
exec login
```

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password: Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second “dialup” password. This will occur only for dial-up connections, and will be prompted by the message “dialup password:”. Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure */etc/profile* is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter [usually *sh(1)*] is initialized, and the file *.profile* in the working directory is executed, if it exists. These specifications are found in the */etc/passwd* file entry for the user. The name of the command interpreter is — followed by the last component of the interpreter’s path name (i.e., *-sh*). If this field in the password file is empty, then the default command interpreter, */bin/sh* is used. If this field is “*”, then a *chroot(2)* is done to the directory named in the directory field of the entry. At that point *login* is re-executed at the new level which must have its own root structure, including */etc/login* and */etc/passwd*.

The basic *environment* [see *environ(5)*] is initialized to:

```
HOME=your-login-directory
PATH=:/bin:/usr/bin
SHELL=last-field-of-passwd-entry
MAIL=/usr/mail/your-login-name
TZ=timezone-specification
```

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your login name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as

```
LN=xxx
```

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables *PATH* and *SHELL* cannot be changed. This prevents people, logging into restricted shell

environments, from spawning secondary shells which are not restricted. Both *login* and *getty* understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it, and allows the inclusion of such things as spaces and tabs.

FILES

<i>/etc/utmp</i>	accounting
<i>/etc/wtmp</i>	accounting
<i>/usr/mail/your-name</i>	mailbox for user <i>your-name</i>
<i>/etc/motd</i>	message-of-the-day
<i>/etc/passwd</i>	password file
<i>/etc/profile</i>	system profile
<i>.profile</i>	user's login profile

SEE ALSO

mail(1), *newgrp*(1), *sh*(1), *su*(1).
passwd(4), *profile*(4), *environ*(5) in the Software Development System manual.

DIAGNOSTICS

Login incorrect if the user name or the password cannot be matched.

No shell, cannot open password file, or no directory: consult a UNIX system programming counselor.

No utmp entry. You must exec "login" from the lowest level "sh". if you attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

NAME

`lp`, `cancel` - send/cancel requests to an LP line printer

SYNOPSIS

`lp` [-c] [-ddest] [-m] [-nnumber] [-ooption] [-s] [-ttitle] [-w] files
`cancel` [ids] [printers]

DESCRIPTION

Lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name - stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

Lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see *cancel*) or find the status [see *lpstat(1)*] of the request.

The following options to *lp* may appear in any order and may be intermixed with file names:

- c Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the -c option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that in the absence of the -c option, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.
- ddest Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted [see *accept(1M)* and *lpstat(1)*]. By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems [see *lpstat(1)*].
- m Send mail [see *mail(1)*] after the files have been printed. By default, no mail is sent upon normal completion of the print request.
- nnumber Print *number* copies (default of 1) of the output.
- ooption Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the -o keyletter more than once. For more information about what is valid for *options*, see *Models* in *lpadmin(1M)*.
- s Suppress messages from *lp(1)* such as "request id is ...".
- ttitle Print *title* on the banner page of the output.
- w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

Cancel cancels line printer requests that were made by the *lp(1)* command. The command line arguments may be either request *ids* [as

printer cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1).
accept(1M), lpadmin(1M), lpsched(1M).

LPSET,LPGET(1M)

NAME

lpset, lpget— initialize the parallel printer driver

SYNOPSIS

lpget device
lpset device indentation columns lines [transparency]

DESCRIPTION

These utilities are used to get and set the current values in the LP driver that define the following:

column indent
columns per line
lines per page
transparency on or off

Transparency mode prevents the LP driver from interpreting control characters and non-printing characters for graphics printers. Any value in the [transparency] field sets transparency on.

EXAMPLE

lpset /dev/lp 0 132 66
would set the values in the driver to their default settings.
lpset /dev/lp 0 132 66 on sets transparency on
lpset /dev/lp 0 13266 off sets transparency off

FILES

/dev/lp

SEE ALSO

lp(7) in the Runtime manual.

NAME
lpstat — print LP status information

SYNOPSIS
lpstat [options]

DESCRIPTION

lpstat prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp(1)* by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

—u"user1, user2, user3"

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

lpstat —o

prints the status of all output requests.

—a[*list*] Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

—c[*list*] Print class names and their members. *List* is a list of class names.

—d Print the system default destination for *lp*.

—o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.

—p[*list*] Print the status of printers. *List* is a list of printer names.

—r Print the status of the LP request scheduler

—s Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

—t Print all status information.

—u[*list*] Print status of output requests for users. *List* is a list of login names.

—v[*list*] Print the names of printers and the path names of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1).

This page intentionally left blank.

NAME

ls - list contents of directory

SYNOPSIS

ls [-RadCxmlnogrtpFbqisf] [names]

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately; but file arguments appear before directories and their contents.

There are three major listing formats. The default format is to list one entry per line, the **-C** and **-x** options enable multicolumn formats, and the **-m** option enables stream output format in which files are listed across the page, separated by commas. In order to determine output formats for the **-C**, **-x**, and **-m** options, *ls* uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the *terminfo* data base is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

There are an unbelievable number of options:

- R** Recursively list subdirectories encountered.
- a** List all entries; usually entries whose names begin with a period (.) are not listed.
- d** If an argument is a directory, list only its name (not its contents); often used with **-l** to get the status of a directory.
- C** Multicolumn output with entries sorted down the columns.
- x** Multicolumn output with entries sorted across rather than down the page.
- m** Stream output format.
- l** List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size.
- n** The same as **-l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.
- o** The same as **-l**, except that the group is not printed.
- g** The same as **-l**, except that the owner is not printed.
- r** Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.
- t** Sort by time modified (latest first) instead of by name.
- u** Use time of last access instead of last modification for sorting (with the **-t** option) or printing (with the **-l** option).
- c** Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**-t**) or printing (**-l**).
- p** Put a slash (/) after each file name if that file is a directory.
- F** Put a slash (/) after each file name if that file is a directory and put an asterisk (*) after each file name if that file is executable.

- b Force printing of non-graphic characters to be in the octal \ddd notation.
- q Force printing of non-graphic characters in file names as the character (?).
- i For each file, print the i-number in the first column of the report.
- s Give size in blocks, including indirect blocks, for each entry.
- f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off -l, -t, -s, and -r, and turns on -a; the order is the order in which entries appear in the directory.

The mode printed under the -l option consists of 10 characters that are interpreted as follows:

The first character is:

- d if the entry is a directory,
- b if the entry is a block special file;
- c if the entry is a character special file;
- p if the entry is a fifo (a.k.a. "named pipe") special file;
- if the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

- r if the file is readable;
- w if the file is writable;
- x if the file is executable;
- if the indicated permission is *not* granted.

The group-execute permission character is given as s if the file has set-group-ID mode; likewise, the user-execute permission character is given as S if the file has set-user-ID mode. The last character of the mode (normally x or -) is t if the 1000 (octal) bit of the mode is on; see *chmod(1)* for the meaning of this mode. The indications of set-ID and 1000 bits of the mode are capitalized (S and T respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

- /etc/passwd to get user IDs for ls -l and ls -o.
- /etc/group to get group IDs for ls -l and ls -g.
- /usr/lib/terminfo/* to get terminal information.

SEE ALSO

chmod(1), find(1).

BUGS

Unprintable characters in file names may confuse the columnar output options.

define	the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $\$n$ in the replacement text, where n is a digit, is replaced by the n -th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; $\$\#$ is replaced by the number of arguments; $\$*$ is replaced by a list of all the arguments separated by commas; $\$@$ is like $\$*$, but each argument is quoted (with the current quotes).
undefine	removes the definition of the macro named in its argument.
defn	returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.
pushdef	like <i>define</i> , but saves any previous definition.
popdef	removes current definition of its argument(s), exposing the previous one, if any.
ifdef	if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word <i>unix</i> is predefined on UNIX system versions of <i>m4</i> .
shift	returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.
changequote	change quote symbols to the first and second arguments. The symbols may be up to five characters long. <i>Changequote</i> without arguments restores the original values (i.e., ` ').
changecom	change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.
divert	<i>m4</i> maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The <i>divert</i> macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.
undivert	causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.
divnum	returns the value of the current output stream.
dnl	reads and discards characters up to and including the next new-line.
ifelse	has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.
incr	returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

<code>decr</code>	returns the value of its argument decremented by 1.
<code>eval</code>	evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> , <code>^</code> (exponentiation), bitwise <code>&</code> , <code> </code> , <code>^</code> , and <code>~</code> ; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
<code>len</code>	returns the number of characters in its argument.
<code>index</code>	returns the position in its first argument where the second argument begins (zero origin), or <code>-1</code> if the second argument does not occur.
<code>substr</code>	returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
<code>translit</code>	transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
<code>include</code>	returns the contents of the file named in the argument.
<code>sinclude</code>	is identical to <code>include</code> , except that it says nothing if the file is inaccessible.
<code>syscmd</code>	executes the UNIX system command given in the first argument. No value is returned.
<code>sysval</code>	is the return code from the last call to <code>syscmd</code> .
<code>maketemp</code>	fills in a string of <code>XXXXX</code> in its argument with the current process ID.
<code>m4exit</code>	causes immediate exit from <code>m4</code> . Argument 1, if given, is the exit code; the default is 0.
<code>m4wrap</code>	argument 1 will be pushed back at final EOF; example: <code>m4wrap('cleanup')</code>
<code>errprint</code>	prints its argument on the diagnostic output file.
<code>dumpdef</code>	prints current names and definitions, for the named items, or for all if no arguments are given.
<code>traceon</code>	with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
<code>traceoff</code>	turns off trace globally and for any macros specified. Macros specifically traced by <code>traceon</code> can be untraced only by specific calls to <code>traceoff</code> .

SEE ALSO

`cc(1)`, `cpp(1)`.

The M4 Macro Processor by B. W. Kernighan and D. M. Ritchie.

MACHID(1)

NAME

pdp11, u3b, u3b5, vax, iAPX286 — provide truth value about your processor type

SYNOPSIS

pdp11

u3b

u3b5

vax

iAPX286

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

- pdp11** True if you are on a PDP-11/45 or PDP-11/70.
- u3b** True if you are on a 3B20 computer.
- u3b5** True if you are on a 3B5 computer.
- vax** True if you are on a VAX-11/750 or VAX-11/780.
- iAPX286** True if you are on an iAPX286 processor.

The commands that do not apply will return a false (non-zero) value. These commands are often used within *make(1)* makefiles and shell procedures to increase portability.

SEE ALSO

make(1), *sh(1)*, *test(1)*, *true(1)*.

NAME

mail, rmail — send mail to users or read mail

SYNOPSIS

```
mail [ -epqr ] [ -f file ]
mail [ -t ] persons
rmail [ -t ] persons
```

DESCRIPTION

Mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a `?`, and a line is read from the standard input to determine the disposition of the message:

<code><new-line></code>	Go on to next message.
<code>+</code>	Same as <code><new-line></code> .
<code>d</code>	Delete message and go on to next message.
<code>p</code>	Print message again.
<code>-</code>	Go back to previous message.
<code>s [files]</code>	Save message in the named <i>files</i> (mbox is default).
<code>w [files]</code>	Save message, without its header, in the named <i>files</i> (mbox is default).
<code>m [persons]</code>	Mail the message to the named <i>persons</i> (yourself is default).
<code>q</code>	Put undeleted mail back in the <i>mailfile</i> and stop.
<code>EOT (control-d)</code>	Same as <code>q</code> .
<code>x</code>	Put all mail back in the <i>mailfile</i> unchanged and stop.
<code>!command</code>	Escape to the shell to do <i>command</i> .
<code>.</code>	Print a command summary.

The optional arguments alter the printing of the mail:

- `-e` causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.
- `-p` causes all mail to be printed without prompting for disposition.
- `-q` causes *mail* to terminate after interrupts. Normally an interrupt only causes the termination of the message being printed.
- `-r` causes messages to be printed in first-in, first-out order.
- `-f file` causes *mail* to use *file* (e.g., **mbox**) instead of the default *mailfile*.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a `.`) and adds it to each *person's mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e., "From ...") are preceded with a `>`. The `-t` option causes the message to be preceded by all *persons* the *mail* is sent to. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the file *dead.letter* will be saved to allow editing and resending. Note that this is regarded as a temporary file in that it is recreated every time needed, erasing the previous contents of *dead.letter*.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark [see *uucp*(1C)]. Everything after the first exclamation mark in *persons* is interpreted by the remote system. In particular, if *persons* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying `a!b!cde` as a recipient's name causes the message to be sent to user `b!cde` on system `a`. System `a` will interpret that destination as a request to send the message to user `cde` on system `b`. This might be useful, for instance, if the sending system can access system `a` but not system `b`, and

MAIL(1)

system **a** has access to system **b**. *Mail* will not use *uucp* if the remote system is the local system name (i.e., `localsystem!user`).

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment. In order for forwarding to work properly the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

Rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

FILES

<code>/etc/passwd</code>	to identify sender and locate persons
<code>/usr/mail/user</code>	incoming mail for <i>user</i> ; i.e., the <i>mailfile</i>
<code>\$HOME/mbox</code>	saved mail
<code>\$MAIL</code>	variable containing path name of <i>mailfile</i>
<code>/tmp/ma*</code>	temporary file
<code>/usr/mail/*.lock</code>	lock for mail directory
<code>dead.letter</code>	unmailable text

SEE ALSO

`login(1)`, `mailx(1)`, `uucp(1C)`, `write(1)`.

BUGS

Conditions sometimes result in a failure to remove a lock file.

After an interrupt, the next message may not be printed; printing may be forced by typing a `p`.

NAME

mailx — interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Incoming mail is stored in a standard file for each user, called the system *mailbox* for that user. When *mailx* is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user's HOME directory [see "MBOX" (ENVIRONMENT VARIABLES) for a description of this file]. Messages remain in this file until forcibly removed.

On the command line, *options* start with a dash (—) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* will attempt to read messages from the *mailbox*. Command line options are:

- d Turn on debugging output. Neither particularly interesting nor recommended.
- e Test for presence of mail. *Mailx* prints nothing and exits with a successful return code if there is mail to read.
- f [*filename*] Read messages from *filename* instead of *mailbox*. If no *filename* is specified, the *mbox* is used.
- F Record the message in a file named after the first recipient. Overrides the "record" variable, if set (see ENVIRONMENT VARIABLES).
- b *number* The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops.
- H Print header summary only.
- i Ignore interrupts. See also "ignore" (ENVIRONMENT VARIABLES).
- n Do not initialize from the system default *Mailx.rc* file.
- N Do not print initial header summary.
- r *address* Pass *address* to network delivery software. All tilde commands are disabled.
- s *subject* Set the Subject header field to *subject*.
- u *user* Read *user's mailbox*. This is only effective if *user's mailbox* is not read protected.
- U Convert *uucp* style addresses to internet standards. Overrides the "conv" environment variable.

When reading mail, *mailx* is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see COMMANDS below). When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by

beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the set and unset commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp(1)* for recording outgoing mail on paper. Alias groups are set by the alias command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, print is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a 'current' message, marked by a '>' in the header summary. Many commands take an optional list of messages (*msglist*) to operate on, which defaults to the current message. A *msglist* is a list of message specifications separated by spaces, which may include:

n	Message number n .
.	The current message.
^	The first undeleted message.
\$	The last message.
*	All messages.
n-m	An inclusive range of message numbers.
user	All messages from user .
/string	All messages with string in the subject line (case ignored).
:c	All messages of type c , where c is one of:
	d deleted messages
	n new messages
	o old messages
	r read messages
	u unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions [see *sh(1)*]. Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* reads commands from a system-wide file (*/usr/lib/mailx/mailx.rc*) to initialize certain parameters, then from a private start-up file (*\$HOME/.mailrc*) for personalized variables. Most regular commands are legal inside start-up files, the most common use being to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, **Copy**, **edit**, **followup**, **Followup**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, and **visual**. Any errors in the start-up file cause the remaining

lines in the file to be ignored.

COMMANDS

The following is a complete list of *mailx* commands:

!shell-command

Escape to the shell. See "SHELL" (ENVIRONMENT VARIABLES).

comment

Null command (comment). This may be useful in *.mailrc* files.

-

Print the current message number:

?

Prints a summary of commands.

alias alias name ...

group alias name ...

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

alternates name ...

Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, *alternates* prints the current list of alternate names. See also "allnet" (ENVIRONMENT VARIABLES).

cd [directory]

chdir [directory]

Change directory. If *directory* is not specified, \$HOME is used.

copy [filename]

copy [msglist] filename

Copy messages to the file without marking the messages as saved. Otherwise equivalent to the *save* command.

Copy [msglist]

Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the *Save* command.

delete [msglist]

Delete messages from the *mailbox*. If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES).

discard [header-field ...]

ignore [header-field ...]

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The *Print* and *Type* commands override this command.

dp [msglist]

dt [msglist]

Delete the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a *delete*

MAILX(1)

command followed by a print command.

echo *string* ...

Echo the given strings [like *echo* (1)].

edit [*msglist*]

Edit the given messages. The messages are placed in a temporary file and the "EDITOR" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES). Default editor is *ed*(1).

exit

xit

Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also *quit*).

file [*filename*]

folder [*filename*]

Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

% the current *mailbox*.

%*user* the *mailbox* for *user*.

the previous file.

& the current *mbox*.

Default file is the current *mailbox*.

folders

Print the names of the files in the directory set by the "folder" variable (see ENVIRONMENT VARIABLES).

followup [*message*]

Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the "record" variable, if set. See also the Followup, Save, and Copy commands and "outfolder" (ENVIRONMENT VARIABLES).

Followup [*msglist*]

Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the followup, Save, and Copy commands and "outfolder" (ENVIRONMENT VARIABLES).

from [*msglist*]

Prints the header summary for the specified messages.

group *alias name* ...

alias *alias name* ...

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *mailrc* file.

headers [*message*]

Prints the page of headers which includes the message specified. The "screen" variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the *z* command.

help

Prints a summary of commands.

hold [*msglist*]**preserve** [*msglist*]Holds the specified messages in the *mailbox*.**if** *s**mail-commands***else***mail-commands***endif**

Conditional execution, where *s* will execute following *mail-commands*, up to an **else** or **endif**, if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the *mailrc* file.

ignore *header-field* ...**discard** *header-field* ...

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The **Print** and **Type** commands override this command.

list

Prints all commands available. No explanation is given.

mail *name* ...

Mail a message to the specified users.

mbox [*msglist*]

Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See "MBOX" (ENVIRONMENT VARIABLES) for a description of this file. See also the **exit** and **quit** commands.

next [*message*]

Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*][*msglist*] [*shell-command*]

Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

preserve [*msglist*]**hold** [*msglist*]Preserve the specified messages in the *mailbox*.

Print [*msglist*]

Type [*msglist*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the `ignore` command.

print [*msglist*]

type [*msglist*]

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is `pg(1)` (see ENVIRONMENT VARIABLES).

quit

Exit from *mailx*, storing messages that were read in *mbox* and unread messages in the *mailbox*. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]

Respond [*msglist*]

Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If "record" is set to a filename, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

reply [*message*]

respond [*message*]

Reply to the specified message, including all other recipients of the message. If "record" is set to a filename, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

Save [*msglist*]

Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the `Copy`, `followup`, and `Followup` commands and "outfolder" (ENVIRONMENT VARIABLES).

save [*filename*]

save [*msglist*] *filename*

Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when *mailx* terminates unless "keepsave" is set (see also ENVIRONMENT VARIABLES and the `exit` and `quit` commands).

set

set *name*

set *name*=*string*

set *name*=*number*

Define a variable called *name*. The variable may be given a null, string, or numeric value. `Set` by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the *mailx* variables.

shell

Invoke an interactive shell [see also "SHELL" (ENVIRONMENT VARIABLES)].

size [*msglist*]

Print the size in characters of the specified messages.

source *filename*

Read commands from the given file and return to command mode.

top [*msglist*]

Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

touch [*msglist*]

Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the *mbox* upon normal termination. See exit and quit.

Type [*msglist*]

Print [*msglist*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the ignore command.

type [*msglist*]

print [*msglist*]

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg(1)* (see ENVIRONMENT VARIABLES).

undelete [*msglist*]

Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

unset *name* ...

Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version

Prints the current version and release date.

visual [*msglist*]

Edit the given messages with a screen editor. The messages are placed in a temporary file and the "VISUAL" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

write [*msglist*] *filename*

Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the save command.

xit

exit

Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also quit).

MAILX(1)

z[+|-]

Scroll the header display forward or backward one screenful. The number of headers displayed is set by the "screen" variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (~). See "escape" (ENVIRONMENT VARIABLES) for changing this special character.

~ shell-command

Escape to the shell.

~.

Simulate end of file (terminate message input).

~: mail-command

~_ mail-command

Perform the command-level request. Valid only when sending a message while reading mail.

~?

Print a summary of tilde escapes.

~A

Insert the autograph string "Sign" into the message (see ENVIRONMENT VARIABLES).

~a

Insert the autograph string "sign" into the message (see ENVIRONMENT VARIABLES).

~b name ...

Add the *names* to the blind carbon copy (Bcc) list.

~c name ...

Add the *names* to the carbon copy (Cc) list.

~d

Read in the *dead.letter* file. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~e

Invoke the editor on the partial message. See also "EDITOR" (ENVIRONMENT VARIABLES).

~f [msglist]

Forward the specified messages. The messages are inserted into the message, without alteration.

~b

Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~i string

Insert the value of the named variable into the text of the message. For example, ~A is equivalent to ~i Sign.

m [*msglist*]

Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

p

Print the message being entered.

q

Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in *dead.letter*. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

r *filename*

< *filename*

< *!shell-command*

Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

s *string ...*

Set the subject line to *string*.

t *name ...*

Add the given *names* to the To list.

y

Invoke a preferred screen editor on the partial message. See also "VISUAL" (ENVIRONMENT VARIABLES).

w *filename*

Write the partial message onto the given file, without the header.

x

Exit as with **q** except the message is not saved in *dead.letter*.

| *shell-command*

Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within *mailx*.

HOME=*directory*

The user's base of operations.

MAILRC=*filename*

The name of the start-up file. Default is \$HOME/.mailrc.

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the set command at any time. The **unset** command may be used to erase variables.

allnet

All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to

behave similarly. Default is **noallnet**. See also the **alternates** command and the "metoo" variable.

append

Upon termination, append messages to the end of the *mbx* file instead of prepending them. Default is **noappend**.

askcc

Prompt for the Cc list after message is entered. Default is **noaskcc**.

asksub

Prompt for subject if it is not specified on the command line with the **-s** option. Enabled by default.

autoprint

Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint**.

bang

Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(1). Default is **nobang**.

cmd=shell-command

Set the default command for the **pipe** command. No default value.

conv=conversion

Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also "sendmail" and the **-U** command line option.

crt=number

Pipe messages having more than *number* lines through the command specified by the value of the "PAGER" variable [*pg*(1) by default]. Disabled by default.

DEAD=filename

The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is **\$HOME/dead.letter**.

debug

Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

dot

Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot**.

EDITOR=shell-command

The command to run when the **edit** or **~e** command is used. Default is *ed*(1).

escape=c

Substitute *c* for the **~** escape character.

folder=directory

The directory for saving standard mail files. User specified file names beginning with a plus (+) are expanded by preceding the filename with this directory name to obtain the real filename. If *directory* does not start with a slash (/), \$HOME is prepended to it. In order to use the plus (+) construct on a *mailx* command line, "folder" must be an exported *sh* environment variable. There is no default for the "folder" variable. See also "outfolder" below.

header

Enable printing of the header summary when entering *mailx*. Enabled by default.

hold

Preserve all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. Default is **nohold**.

ignore

Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof

Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the `^.` command. Default is **noignoreeof**. See also "dot" above.

keep

When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave

Keep messages that have been saved in other files in the *mailbox* instead of deleting them. Default is **nokeepsave**.

MBOX=filename

The name of the file to save messages which have been read. The *xit* command overrides this function, as does saving the message explicitly in another file. Default is \$HOME/mbox.

metoo

If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

LISTER=shell-command

The command (and options) to use when listing the contents of the "folder" directory. The default is *ls(1)*.

onehop

When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder

Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the path name is absolute. Default is **nooutfolder**. See "folder" above and the Save, Copy, followup, and Followup commands.

page

Used with the pipe command to insert a form feed after each message sent through the pipe. Default is **no page**.

PAGER=shell-command

The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg(1)*.

prompt=string

Set the *command mode* prompt to *string*. Default is "? ".

quiet

Refrain from printing the opening message and version when entering *mailx*. Default is **noquiet**.

record=filename

Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save

Enable saving of messages in *dead.letter* on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen=number

Sets the number of lines in a screen full of headers for the headers command.

sendmail=shell-command

Alternate command for delivering messages. Default is *mail(1)*.

sendwait

Wait for background mailer to finish before returning. Default is **nosendwait**.

SHELL=shell-command

The name of a preferred command interpreter. Default is *sh(1)*.

showto

When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=string

The variable inserted into the text of a message when the `~a` (autograph) command is given. No default [see also `~i` (TILDE ESCAPES)].

Sign=string

The variable inserted into the text of a message when the `~A` command is given. No default [see also `~i` (TILDE ESCAPES)].

toplines=number

The number of lines of header to print with the **top** command. Default is 5.

VISUAL=shell-command

The name of a preferred screen editor. Default is **vi(1)**.

FILES

\$HOME/.mailrc	personal start-up file
\$HOME/mbox	secondary storage file
/usr/mail/*	post office directory
/usr/lib/mailx/mailx.help*	help message files
/usr/lib/mailx/mailx.rc	global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO

mail(1), **pg(1)**, **ls(1)**.

BUGS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

The full internet addressing is not fully supported by *mailx*. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a "." are treated as the end of the message by *mail(1)* (the standard mail delivery program).

MAKE(1)

NAME

make — maintain, update, and regenerate groups of programs

SYNOPSIS

make [**-f** *makefile*] [**-p**] [**-i**] [**-k**] [**-s**] [**-r**] [**-n**] [**-b**] [**-e**] [**-m**]
[**-t**] [**-d**] [**-q**] [*names*]

DESCRIPTION

The following is a brief description of all options and some special names:

- f** *makefile* Description file name. *Makefile* is assumed to be the name of a description file. A file name of **-** denotes the standard input. The contents of *makefile* override the built-in rules if they are present.
- p** Print out the complete set of macro definitions and target descriptions.
- i** Ignore error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.
- k** Abandon work on the current entry, but continue on other branches that do not depend on that entry.
- s** Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.
- r** Do not use the built-in rules.
- n** No execute mode. Print commands, but do not execute them. Even lines beginning with an **@** are printed.
- b** Compatibility mode for old makefiles.
- e** Environment variables override assignments within makefiles.
- m** Print a memory map showing text, data, and stack. This option is a no-operation on systems without the *getu* system call.
- t** Touch the target files (causing them to be up-to-date) rather than issue the usual commands.
- d** Debug mode. Print out detailed information on files and times examined.
- q** Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.
- .DEFAULT** If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.
- .PRECIOUS** Dependents of this target will not be removed when quit or interrupt are hit.
- .SILENT** Same effect as the **-s** option.
- .IGNORE** Same effect as the **-i** option.

Make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **-f** option is present, **makefile**, **Makefile**, **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is **-**, the standard input is taken. More than one **-** *makefile* argument pair may appear.

Make updates a target only if its dependents are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a `;`, then a (possibly null) list of prerequisite files or dependencies. Text following a `;` and all following lines that begin with a tab are shell commands to be executed to update the target. The first line that does not begin with a tab or `#` begins a new dependency or macro definition. Shell commands may be continued across lines with the `<backslash><new-line>` sequence. Everything printed by `make` (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a \
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (`#`) and new-line surround comments.

The following *makefile* says that `pgm` depends on two files `a.o` and `b.o`, and that they in turn depend on their corresponding source files (`a.c` and `b.c`) and a common file `incl.b`:

```
pgm: a.o b.o
    cc a.o b.o -o pgm
a.o: incl.h a.c
    cc -c a.c
b.o: incl.h b.c
    cc -c b.c
```

Command lines are executed one at a time, each by its own shell. The first one or two characters in a command can be the following: `-`, `@`, `-@`, or `@-`. If `@` is present, printing of the command is suppressed. If `-` is present, `make` ignores an error. A line is printed when it is executed unless the `-s` option is present, or the entry `.SILENT:` is in *makefile*, or unless the initial character sequence contains a `@`. The `-n` option specifies printing without execution; however, if the command line has the string `$(MAKE)` in it, the line is always executed (see discussion of the `MAKEFLAGS` macro under *Environment*). The `-t` (`touch`) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate `make`. If the `-i` option is present, or the entry `.IGNORE:` appears in *makefile*, or the initial character sequence of the command contains `-.` the error is ignored. If the `-k` option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The `-b` option allows old *makefiles* (those written for the old version of `make`) to run without errors. The difference between the old version of `make` and this version is that this version requires all dependency lines to have a (possibly null or implicit) command associated with them. The previous version of `make` assumed, if no command was specified explicitly, that the command was null.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name `.PRECIOUS`.

Environment

The environment is read by `make`. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any *makefile* and after the internal rules; thus, macro assignments in a *makefile* override environment variables. The `-e` option causes the environment to override the macro assignments in a *makefile*.

The MAKEFLAGS environment variable is processed by *make* as containing any legal input option (except *-f*, *-p*, and *-d*) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, MAKEFLAGS always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the *-n* option is used, the command \$(MAKE) is executed anyway; hence, one can perform a *make -n* recursively on a whole software system to see what would have been executed. This is because the *-n* is put in MAKEFLAGS and passed to further invocations of \$(MAKE). This is one way of debugging all of the makefiles for a software project without actually doing anything.

Macros

Entries of the form *string1 = string2* are macro definitions. *String2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of \$(*string1*[:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional *:subst1=subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

- \$• The macro \$• stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.
- \$@ The \$@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.
- \$< The \$< macro is only evaluated for inference rules or the .DEFAULT rule. It is the module which is out-of-date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the .c.o rule, the \$< macro would evaluate to the .c file. An example for making optimized .o files from .c files is:

```
.c.o:
    cc -c -O $•.c
```

or:

```
.c.o:
    cc -c -O $<
```

- \$? The \$? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules which must be rebuilt.
- \$% The \$% macro is only evaluated when the target is an archive library member of the form lib(file.o). In this case, \$@ evaluates to lib and \$% evaluates to the library member, file.o.

Four of the five macros can have alternative forms. When an uppercase D or F is appended to any of the four macros, the meaning is changed to "directory part" for D and "file part" for F. Thus, \$(@D) refers to the directory part of the string \$@. If there is no directory part, ./ is generated. The only macro excluded from this alternative form is \$?. The reasons for this are debatable.

Suffixes

Certain names (for instance, those ending with `.o`) have inferable prerequisites such as `.c`, `.s`, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c~ .sh .sh~ .c.o .c~.o .c~.c .s.o .s~.o .y.o .y~.o .l.o .l~.o
.y.c .y~.c .l.c .c.a .c~.a .s~.a .h~.h
```

The internal rules for *make* are contained in the source file `rules.c` for the *make* program. These rules can be locally modified. To print out the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used:

```
make -fp - 2>/dev/null </dev/null
```

The only peculiarity in this output is the `(null)` string which *printf*(3S) prints when handed a null string.

A tilde in the above rules refers to an SCCS file [see *sccsfile*(4)]. Thus, the rule `.c~.o` would transform an SCCS C source file into an object file (`.o`). Because the `s.` of the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., `.c:`) is the definition of how to build `x` from `x.c`. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for `.SUFFIXES`. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

```
.SUFFIXES: .o .c .y .l .s
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; `.SUFFIXES:` with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

```
pgm: a.o b.o
      cc a.o b.o -o pgm
a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, `CFLAGS`, `LFLAGS`, and `YFLAGS` are used for compiler options to `cc(1)`, `lex(1)`, and `yacc(1)`, respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix `.o` from a file with suffix `.c` is specified as an entry with `.c.o:` as the target and no dependents. Shell commands associated with the target define the rule for making a `.o` file from a `.c` file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

MAKE(1)

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus `lib(file.o)` and `$(LIB)(file.o)` both refer to an archive library which contains `file.o`. (This assumes the `LIB` macro has been previously defined.) The expression `$(LIB)(file1.o file2.o)` is not legal. Rules pertaining to archive libraries have the form `.XX.a` where the `XX` is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the `XX` to be different from the suffix of the archive member. Thus, one cannot have `lib(file.o)` depend upon `file.o` explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date

.c.a:
        $(CC) -c $(CFLAGS) $<
        ar rv $@ $*.o
        rm -f $*.o
```

In fact, the `.c.a` rule listed above is built into `make` and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        ar rv lib $?
        rm $? @echo lib is now up-to-date

.c.a;;
```

Here the substitution mode of the macro expansions is used. The `$?` list is defined to be the set of object file names (inside `lib`) whose C source files are out-of-date. The substitution mode translates the `.o` to `.c`. (Unfortunately, one cannot as yet transform to `.c`; however, this may become possible in the future.) Note also, the disabling of the `.c.a` rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

FILES

[Mm]akefile and s.[Mm]akefile

SEE ALSO

`cc(1)`, `cd(1)`, `lex(1)`, `sh(1)`, `yacc(1)`,
`printf(3S)`, `scscfile(4)` in the Software Development System manual.

BUGS

Some commands return non-zero status inappropriately; use `-i` to overcome the difficulty. File names with the characters `= : @` will not work. Commands that are directly executed by the shell, notably `cd(1)`, are ineffectual across new-lines in `make`. The syntax `lib(file1.o file2.o file3.o)` is illegal. You cannot build `lib(file.o)` from `file.o`. The macro `$(a:.o=.c)` does not work.

NAME man - print on-line documentation

SYNOPSIS
man *command*

DESCRIPTION
Man is a shell command file which prints on-line documentation for the commands.

DIAGNOSTICS
"man: command not found"

FILES
/usr/catman/?_man/man[1-8]/*
preformatted manual entries

MESG(1)

NAME

mesg - permit or deny messages

SYNOPSIS

mesg [*n*] [*y*]

DESCRIPTION

Mesg with argument *n* forbids messages via *write*(1) by revoking nonuser write permission on the user's terminal. *Mesg* with argument *y* reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

`mkdir` - make a directory

SYNOPSIS

`mkdir` *dirname* ...

DESCRIPTION

Mkdir creates specified directories in mode 777 (possibly altered by *umask*(1)). Standard entries, `..`, for the directory itself, and `..`, for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

sh(1), *rm*(1), *umask*(1).

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

MKFS(1M)

NAME

mkfs — construct a file system

SYNOPSIS

/etc/mkfs special blocks[:i-nodes] [gap blocks/cyl]
/etc/mkfs special proto [gap blocks/cyl]

DESCRIPTION

Mkfs constructs a file system by writing on the special file according to the directions found in the remainder of the command line. The command waits 10 seconds before starting to construct the file system. If the second argument is given as a string of digits, *mkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* disk blocks the file system will occupy. The boot program is left uninitialized. If the optional number of i-nodes is not given, the default is the number of *logical* blocks divided by 4.

If the second argument is a file name that can be opened, *mkfs* assumes it to be a prototype file *proto*, and will take its directions from that file. The prototype file contains tokens separated by spaces or new-lines. The first token is the name of a file to be copied onto block zero as the bootstrap program. The second token is a number specifying the size of the created file system in *physical* disk blocks. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of i-nodes in the file system. The maximum number of i-nodes configurable is 65500. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6-character string. The first character specifies the type of the file. (The characters *-bcd* specify regular, block special, character special and directory files respectively.) The second character of the type is either *u* or *-* to specify set-user-id mode or not. The third is *g* or *-* for the set-group-id mode. The rest of the mode is a 3-digit octal number giving the owner, group, and other read, write, execute permissions [see *chmod*(1)].

Two decimal number tokens come after the mode; they specify the user and group IDs of the owner of the file.

If the file is a regular file, the next token is a path name whence the contents and size are copied. If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries *.* and *..* and then reads a list of names and (recursively) files specifications for the entries in the directory. The scan is terminated with the token *\$*.

A sample prototype specification follows:

```
/stand/diskboot
4872 110
d--777 3 1
usr      d--777 3 1
sh       ----755 3 1 /bin/sh
ken      d--755 6 1
          $
b0       b--644 3 1 0 0
c0       c--644 3 1 0 0
          $
$
```


In both command syntaxes, the rotational *gap* and the number of *blocks/cyl* can be specified. The *default* will be used if the supplied *gap* and *blocks/cyl* are considered illegal values or if a short argument count occurs.

SEE ALSO

chmod(1), dir(4), fs(4).

BUGS

If a prototype is used, it is not possible to initialize a file larger than 64K bytes, nor is there a way to specify links.

MKNOD(1M)

NAME

mknod - build special file

SYNOPSIS

```
/etc/mknod name c | b major minor  
/etc/mknod name p
```

DESCRIPTION

Mknod makes a directory entry and corresponding i-node for a special file. The first argument is the *name* of the entry. In the first case, the second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g., unit, drive, or line number), which may be either decimal or octal.

The assignment of major device numbers is specific to each system. They have to be dug out of the system source file **conf.c**.

Mknod can also be used to create fifo's (a.k.a named pipes) (second case in *SYNOPSIS* above).

SEE ALSO

mknod(2).

NAME

mount, umount - mount and dismount file system

SYNOPSIS

/etc/mount [special directory [-r]]

/etc/umount special

DESCRIPTION

Mount announces to the system that a removable file system is present on the device *special*. The *directory* must exist already; it becomes the name of the root of the newly mounted file system.

These commands maintain a table of mounted devices. If invoked with no arguments, *mount* prints the table.

The optional last argument indicates that the file is to be mounted read-only. A file system on a physically write-protected media can only be mounted read-only.

Umount announces to the system that the removable file system previously mounted on device *special* is to be removed.

FILES

/etc/mnttab mount table

SEE ALSO

setmnt(1M), mount(2), mnttab(4).

DIAGNOSTICS

Mount issues a warning if the file system to be mounted is currently mounted under another name.

Umount complains if the special file is not mounted or if it is busy. The file system is busy if it contains an open file or some user's working directory.

MVDIR(1M)

NAME

`mmdir` — move a directory

SYNOPSIS

`/etc/mmdir` *dirname* *name*

DESCRIPTION

Mmdir moves directories within a file system. *Dirname* must be a directory; *name* must not exist. Neither name may be a sub-set of the other (*/x/y* cannot be moved to */x/y/z*, nor vice versa).

Only super-user can use *mmdir*.

SEE ALSO

`mkdir(1)`.

NAME

ncheck - generate names from i-numbers

SYNOPSIS

/etc/ncheck [*-i* numbers] [*-a*] [*-s*] [file-system]

DESCRIPTION

Ncheck with no argument generates a path name vs. i-number list of all files on a set of default file systems. Names of directory files are followed by *./*. The *-i* option reduces the report to only those files whose i-numbers follow. The *-a* option allows printing of the names *.* and *..*, which are ordinarily suppressed. The *-s* option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

A file system may be specified.

The report is in no useful order, and probably should be sorted.

SEE ALSO

fsc(1M), *sort*(1).

DIAGNOSTICS

When the file system structure is improper, *??* denotes the "parent" of a parentless file and a path name beginning with *...* denotes a loop.

NEWFORM(1)

NAME

newform — change the format of a text file

SYNOPSIS

newform [-s] [-itabspec] [-otabspec] [-bn] [-en] [-pn] [-an] [-f]
[-cchar] [-ln] [files]

DESCRIPTION

Newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for **-s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like "**-e15 -l60**" will yield results different from "**-l60 -e15**". Options are applied to all *files* on the command line.

-itabspec Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs*(1). In addition, *tabspec* may be **-**, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input [see *fspec*(4)]. If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** expects no tabs; if any are found, they are treated as **-1**.

-otabspec Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **-itabspec**. If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** means that no spaces will be converted to tabs on output.

-ln Set the effective line length to *n* characters. If *n* is not entered, **-l** defaults to 72. The default line length without the **-l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **-i** to expand tabs to spaces).

-bn Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **-ln**). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **-b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform -ll -b7 file-name
```

The **-ll** must be used to set the effective line length shorter than any existing line in the file so that the **-b** option is activated.

-en Same as **-bn** except that characters are truncated from the end of the line.

-ck Change the prefix/append character to *k*. Default character for *k* is a space.

-pn Prefix *n* characters (see **-ck**) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

-an Same as **-pn** except characters are appended to the end of a line.

-f Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last*

- o** option. If no **-o** option is specified, the line which is printed will contain the default specification of **-8**.
- s** Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.
- An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.
- For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:
- ```
newform -s -i -l -a -e file-name
```

**DIAGNOSTICS**

All diagnostics are fatal.

*usage: ...*

*not -s format*

*can't open file*

*internal line too long*

*tabspec in error*

*tabspec indirection illegal*

*Newform* was called with a bad option.

There was no tab on one line.

Self-explanatory.

A line exceeds 512 characters after being expanded in the internal work buffer.

A tab specification is incorrectly formatted, or specified tab stops are not ascending.

A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

**EXIT CODES**

0 - normal execution

1 - for any error

**SEE ALSO**

*csplit(1)*, *tabs(1)*.

*fspec(4)* in the Software Development System manual.

**BUGS**

*Newform* normally only keeps track of physical characters; however, for the **-i** and **-o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

*Newform* will not prompt the user if a *tabspec* is to be read from the standard input (by use of **-i--** or **-o--**).

If the **-f** option is used, and the last **-o** option specified was **-o--**, and was preceded by either a **-o--** or a **-i--**, the tab specification format line will be incorrect.

## NEWGRP(1)

### NAME

`newgrp` — log in to a new group

### SYNOPSIS

`newgrp` [ - ] [ group ]

### DESCRIPTION

*Newgrp* changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp*, regardless of whether it terminated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. System variables (such as `PS1`, `PS2`, `PATH`, `MAIL`, and `HOME`), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (`PS1`) other than `$` (default) and has not exported `PS1`. After an invocation of *newgrp*, successful or not, their `PS1` will now be set to the default prompt string `$`. Note that the shell command *export* [see *sh*(1)] is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry.

If the first argument to *newgrp* is a `-`, the environment is changed to what would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in `/etc/group` as being a member of that group.

### FILES

|                          |                        |
|--------------------------|------------------------|
| <code>/etc/group</code>  | system's group file    |
| <code>/etc/passwd</code> | system's password file |

### SEE ALSO

`login`(1), `sh`(1),  
`group`(4), `passwd`(4), `environ`(5) in the Software Development System manual.

### BUGS

There is no convenient way to enter a password into `/etc/group`. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.



**NAME**

*news* - print news items

**SYNOPSIS**

*news* [ *-a* ] [ *-n* ] [ *-s* ] [ *items* ]

**DESCRIPTION**

*News* is used to keep the user informed of current events. By convention, these events are described by files in the directory */usr/news*.

When invoked without arguments, *news* prints the contents of all current files in */usr/news*, most recent first, with each preceded by an appropriate header. *News* stores the "currency" time as the modification date of a file named *.news\_time* in the user's home directory (the identity of this directory is determined by the environment variable *\$HOME*); only files more recent than this currency time are considered "current."

The *-a* option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

The *-n* option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

The *-s* option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's *.profile* file or in the system's */etc/profile*.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

**FILES**

*/etc/profile*  
*/usr/news/\**  
*\$HOME/.news\_time*

**SEE ALSO**

*profile*(4), *environ*(5) in the Software Development System manual.

## NICE(1)

### NAME

`nice` - run a command at low priority

### SYNOPSIS

`nice` [ `-increment` ] `command` [ `arguments` ]

### DESCRIPTION

*Nice* executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., `--10`.

### SEE ALSO

`nohup`(1).

`nice`(2) in the Software Development System manual.

### DIAGNOSTICS

*Nice* returns the exit status of the subject command.

### BUGS

An *increment* larger than 19 is equivalent to 19.

**NAME**

nl - line numbering filter

**SYNOPSIS**nl [-h**type**] [-b**type**] [-f**type**] [-v**start#**] [-i**incr**] [-p] [-l**num**] [-s**sep**]  
[-w**width**] [-n**format**] [-d**delim**] *file***DESCRIPTION**

*Nl* reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

*Nl* views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

| <i>Line contents</i> | <i>Start of</i> |
|----------------------|-----------------|
| \\:\:                | header          |
| \\:                  | body            |
| \\:                  | footer          |

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

- btype** Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: **a**, number all lines; **t**, number lines with printable text only; **n**, no line numbering; **pstring**, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is **t** (text lines numbered).
- h**type**** Same as **-btype** except for header. Default *type* for logical page header is **n** (no lines numbered).
- f**type**** Same as **-btype** except for footer. Default for logical page footer is **n** (no lines numbered).
- p** Do not restart numbering at logical page delimiters.
- v**start#**** *Start#* is the initial value used to number logical page lines. Default is **1**.
- i**incr**** *Incr* is the increment value used to number logical page lines. Default is **1**.
- s**sep**** *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.
- w**width**** *Width* is the number of characters to be used for the line number. Default *width* is **6**.
- n**format**** *Format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes suppressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).

## NL(1)

- lnum** *Num* is the number of blank lines to be considered as one. For example, **-l2** results in only the second adjacent blank being numbered (if the appropriate **-ha**, **-ba**, and/or **-fa** option is set). Default is 1.
- dxx** The delimiter characters specifying the start of a logical page section may be changed from the default characters (\;) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **-d** and the delimiter characters. To enter a backslash, use two backslashes.

### EXAMPLE

The command:

```
nl -v10 -i10 -d!+ file1
```

will number *file1* starting at line number 10 with an increment of ten. The logical page delimiters are !+.

### SEE ALSO

`pr(1)`.

**NAME**

**nm** - print name list of common object files

**SYNOPSIS**

**nm** [**options**] file-names

**DESCRIPTION**

The *nm* command displays the symbol table of each common object file *file-name*. *File-name* may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed. Note that the object file must have been compiled with the **-g** option of the *cc(1)* command for there to be **Type**, **Size**, or **Line** information.

**Name** The name of the symbol.

**Value** Its value expressed as an offset or an address depending on its storage class.

**Class** Its storage class.

**Type** Its type and derived type. If the symbol is an instance of a structure or of a union, then the structure or union tag will be given following the type (e.g., struct-tag). If the symbol is an array, then the array dimensions will be given following the type (e.g., **char**[*n*][*m*]).

**Size** Its size in bytes, if available.

**Line** The source line number at which it is defined, if available.

**Section** For storage classes static and external, the object file section containing the symbol (e.g., text, data, or bss).

The output of *nm* may be controlled using the following options:

- d** Print the value and size of a symbol in decimal (the default).
- e** Print only external and static symbols.
- f** Produce full output. Print redundant symbols (.text, .data, and .bss), normally suppressed.
- h** Do not display the output header data.
- n** Sort external symbols by name before they are printed.
- o** Print the value and size of a symbol in octal instead of hexadecimal.
- p** Produce easily parsable, terse output (similar to pre-5.0 nm). Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **S** (user-defined segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (nonexternal) the type letter is in lowercase.
- r** Prepend the name of the object file to each output line.
- u** Print undefined symbols only.
- v** Sort external symbols by value before they are printed.
- x** Print the value and size of a symbol in hexadecimal.
- T** By default, *nm* prints the entire name of the symbols listed. Since object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The **-T** option

## NM(1)

causes *nm* to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

**-V** Print the version of the *nm* command executing on the standard error output.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both **nm name -e -v** and **nm -ve name** print the static and external symbols in *name*, with external symbols sorted by value.

### CAVEATS

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the **-v** and **-n** options should be used only in conjunction with the **-e** option.

### SEE ALSO

*as(1)*, *cc(1)*, *ld(1)*.  
*a.out(4)*, *ar(4)* in the Software Development System manual.

### DIAGNOSTICS

"nm: name: cannot open"  
if *name* cannot be read.

"nm: name: bad magic"  
if *name* is not an appropriate common object file.

"nm: name: no symbols"  
if the symbols have been stripped from *name*.

**NAME**

*/etc/nodename* - change or display system node name

**SYNOPSIS**

*/etc/nodename* [string]

**DESCRIPTION**

*Nodename*, with no parameters, displays the current system node name for *uucp* connections. The default system node name is "system5."

*Nodename string* invokes the *patch* utility in order to change the system node name to "string" in the System V/AT kernel residing in the root directory of the hard disk. The system must be rebooted to activate the new node name, which will then be displayed above the "login:" prompt.

**SEE ALSO**

*usr/include/sys/utsname.h*

## NOHUP(1)

### NAME

`nohup` — run a command immune to hangups and quits

### SYNOPSIS

`nohup` *command* [*arguments* ]

### DESCRIPTION

*Nohup* executes *command* with hangups and quits ignored. If output is not re-directed by the user, both standard output and standard error are sent to `nohup.out`. If `nohup.out` is not writable in the current directory, output is redirected to `$HOME/nohup.out`.

### EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the *nohup* applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored [see *sh*(1)]:

```
nohup file &
```

An example of what the contents of *file* could be is:

```
tbl ofile | eqn | nroff > nfile
```

### SEE ALSO

`chmod`(1), `nice`(1), `sh`(1).

`signal`(2) in the Software Development System manual.

### WARNINGS

`nohup command1; command2` *nohup* applies only to *command1*!

`nohup (command1; command2)` is syntactically incorrect.

Be careful of where standard error is redirected. The following command may put error messages on tape, making it unreadable:

```
nohup cpio -o <list >/dev/rmt/1m&
```

while

```
nohup cpio -o <list >/dev/rmt/1m 2>errors&
```

puts the error messages into file *errors*.



**NAME**

od - octal dump

**SYNOPSIS**

od [ -bcdosx ] [ file ] [ [ + ] offset . ] [ b ] ]

**DESCRIPTION**

*Od* dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, -o is default. The meanings of the format options are:

- b Interpret bytes in octal.
- c Interpret bytes in ASCII. Certain nongraphic characters appear as C escapes: null=\0, backspace=\b, form-feed=\f, new-line=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.
- d Interpret words in unsigned decimal.
- o Interpret words in octal.
- s Interpret 16-bit words in signed decimal.
- x Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If . is appended, the offset is interpreted in decimal. If b is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by +.

Dumping continues until end-of-file.

**SEE ALSO**

dump(1).

# PACK(1)

## NAME

pack, pcat, unpack — compress and expand files

## SYNOPSIS

**pack** [ - ] [ -f ] name ...

**pcat** name ...

**unpack** name ...

## DESCRIPTION

*Pack* attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name.z* with the same access modes, access and modified dates, and owner as those of *name*. The **-f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

*Pack* uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the **-** argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of **-** in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each *.z* file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

*Pack* returns a value that is the number of files that it failed to compress.

No packing will occur if:

- the file appears to be already packed;
- the file name has more than 12 characters;
- the file has links;
- the file is a directory;
- the file cannot be opened;
- no disk storage blocks will be saved by packing;
- a file called *name.z* already exists;
- the *.z* file cannot be created;
- an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended *.z* extension. Directories cannot be compressed.

*Pcat* does for packed files what *cat(1)* does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name.z* use:

pcat name.z

or just:

pcat name

To make an unpacked copy, say *nnn*, of a packed file named *name.z* (without destroying *name.z*) use the command:

```
pcat name >nnn
```

*Pcat* returns the number of files it was unable to unpack. Failure may occur if:

- the file name (exclusive of the *.z*) has more than 12 characters;
- the file cannot be opened;
- the file does not appear to be the output of *pack*.

*Unpack* expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name.z* (or just *name*, if *name* ends in *.z*). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the *.z* suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

*Unpack* returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

- a file with the "unpacked" name already exists;
- if the unpacked file cannot be created.

SEE ALSO  
cat(1).

## PASSWD(1)

### NAME

`passwd` — change login password

### SYNOPSIS

`passwd [ name ]`

### DESCRIPTION

This command changes or installs a password associated with the login *name*.

Ordinary users may change only the password which corresponds to their login *name*.

*passwd* prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered *passwd* checks to see if the old password has “aged” sufficiently. If “aging” is insufficient the new password is rejected and *passwd* terminates; see *passwd*(4).

Assuming “aging” is sufficient, a check is made to insure that the new password meets construction requirements. When the new password is entered a second time the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

- Each password must have at least six characters. Only the first eight characters are significant.

- Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, “alphabetic” means upper and lowercase letters.

- Each password must differ from the user’s login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

- New passwords must differ from the old by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

One whose effective user ID is zero is called a super-user; see *id*(1), and *su*(1). Super-users may change any password; hence, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

### FILES

`/etc/passwd`

### SEE ALSO

*login*(1), *id*(1), *su*(1),  
*crypt*(3C), *passwd*(4) in the Software Development System manual.

**NAME**

passwd - password file

**DESCRIPTION***Passwd* contains for each user the following information:

login name  
 encrypted password  
 numerical user ID  
 numerical group ID  
 GCOS job number, box number, optional GCOS user ID  
 initial working directory  
 program to use as Shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. The GCOS field is used only when communicating with that system, and in other installations can contain any desired information. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, the Shell itself is used.

This file resides in directory */etc*. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user IDs to names.

The encrypted password consists of 13 characters chosen from a 64-character alphabet (*., /, 0-9, A-Z, a-z*), except when the password is null, in which case the encrypted password is also null. Password aging is effected for a particular user if this encrypted password in the passwd file is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced in the first instance by the super-user.)

The first character of the age, *M* say, denotes the maximum number of weeks for which a password is valid. Users who attempt to log in after their passwords have expired will be forced to supply a new one. The next character, *m* say, denotes the minimum period in weeks which must expire before the password may be changed. The remaining characters define the week (counted from the beginning of 1970) when the password was last changed. (A null string is equivalent to zero.) *M* and *m* have numerical values in the range 0-63 that correspond to the 64-character alphabet shown above (i.e., */* = 1 week; *z* = 63 weeks). If *m* = *M* = 0 (derived from the string *.* or *..*) users will be forced to change their passwords the next time they log in (and the "age" will disappear from their entries in the passwd file). If *m* > *M* (signified, e.g., by the string *./*) only the super-user will be able to change the password.

**FILES***/etc/passwd***SEE ALSO**

*a64l(3C)*, *crypt(3C)*, *getpwent(3C)*, *group(4)*,  
*login(1)*, *passwd(1)* in the Runtime System manual.

# PASTE(1)

## NAME

paste — merge same lines of several files or subsequent lines of one file

## SYNOPSIS

```
paste file1 file2 ...
paste -dlist file1 file2 ...
paste -s [-dlist] file1 file2 ...
```

## DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if *-* is used in place of a file name.

The meanings of the options are:

- d** Without this option, the new-line characters of each but the last file (or last line in case of the *-s* option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).
- list* One or more characters immediately following *-d* replace the default *tab* as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no *-s* option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: *\n* (new-line), *\t* (tab), *\\* (backslash), and *\0* (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use *-d"\\\\"*).
- s* Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with *-d* option. Regardless of the *list*, the very last character of the file is forced to be a new-line.
- May be used in place of any file name, to read a line from the standard input. (There is no prompting).

## EXAMPLES

```
ls | paste -d" " - list directory in one column
ls | paste - - - - list directory in four columns
paste -s -d"\t\n" file combine pairs of lines into lines
```

## SEE ALSO

cut(1), grep(1), pr(1).

## DIAGNOSTICS

*line too long*

Output lines are restricted to 511 characters.

*too many files*

Except for *-s* option, no more than 12 input files may be specified.

**NAME**

*patch* - inspect or modify an STL- or COFF-format binary file

**SYNOPSIS**

*patch* [-k] file [-lbisc] [+offset] symbol [datum] [datum]

**DESCRIPTION**

*Patch* displays or modifies parts of an STL or COFF format, o or a.out file. The -k option inspects or modifies /dev/kmem instead of the binary file, and when used with a kernel binary file (usually /system5) may be used to hot-patch the running operating system. When used with no data arguments, *patch* simply prints out the contents of symbol or symbol+offset in the format given by one of -lbisc, which denote long, byte, integer, string, char, respectively. Integer is the default data size. When used with one or more data arguments, *patch* modifies successive locations starting from symbol or symbol+offset with the given data. Long, integer, and byte data are numeric and may be preceded with a 0 or 0x to denote octal or hexadecimal data. A numeric address may be used instead of symbol. Character data may be one character long or in backslash form:

\123 style strings are collapsed into a single character.

\000 denotes 0.

\t gets replaced by a tab

\n gets replaced by a newline

\r gets replaced by a carriage return

\b gets replaced by a backspace

String data may be one or more characters, with the above escape sequences. A null is always placed after each string datum.

## PG(1)

### NAME

`pg` - file perusal filter for soft-copy terminals

### SYNOPSIS

`pg [-number] [-p string] [-cefnus] [+linenumber] [+ /pattern/] [files...]`

### DESCRIPTION

The `pg` command is a filter which allows the examination of *files* one screenful at a time on a soft-copy terminal. (The file name `-` and/or `NULL` arguments indicate that `pg` should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, `pg` scans the *terminfo*(4) data base for the terminal type specified by the environment variable `TERM`. If `TERM` is not defined, the terminal type `dumb` is assumed.

The command line options are:

#### `-number`

An integer specifying the size (in lines) of the window that `pg` is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

#### `-p string`

Causes `pg` to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

#### `-c`

Home the cursor and clear the screen before displaying each page. This option is ignored if `clear_screen` is not defined for this terminal type in the *terminfo*(4) data base.

#### `-e`

Causes `pg` *not* to pause at the end of each file.

#### `-f`

Normally, `pg` splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The `-f` option inhibits `pg` from splitting lines.

#### `-n`

Normally, commands must be terminated by a `<newline>` character. This option causes an automatic end of command as soon as a command letter is entered.

#### `-s`

Causes `pg` to print all messages and prompts in standout mode (usually inverse video).

#### `+linenumber`

Start up at *linenumber*.

#### `+ /pattern/`

Start up at the first line containing the regular expression pattern.

The responses that may be typed when `pg` pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or



line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1) <*newline*> or <*blank*>

This causes one page to be displayed. The address is specified in pages.

(+1) I With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) d or ^D

Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or ^L Typing a single period causes the current page of text to be redisplayed.

\$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed(1)* are available. They must always be terminated by a <*newline*>, even if the *-n* option is specified.

*i/pattern/*

Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i^pattern^*  
*i?pattern?*

Search backward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending *m* or *b* to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix *t* can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

*in* Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*ip* Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*iw* Display another window of text. If *i* is present, set the window size to *i*.

*s filename*

Save the input in the named file. Only the current file being perused is saved. The white space between the *s* and *filename* is optional. This command must always be terminated by a <*newline*>, even if the *-n* option is specified.

*b* Help by displaying an abbreviated summary of available commands.

*q* or *Q* Quit *pg*.

**!command**

*Command* is passed to the shell, whose name is taken from the SHELL environment variable. If this is not available, the default shell is used. This command must always be terminated by a *<newline>*, even if the *-n* option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

**EXAMPLE**

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

**NOTES**

While waiting for terminal input, *pg* responds to BREAK, DEL, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the *z* and *f* commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

**FILES**

|                            |                                          |
|----------------------------|------------------------------------------|
| <i>/usr/lib/terminfo/*</i> | Terminal information data base           |
| <i>/tmp/pg*</i>            | Temporary file when input is from a pipe |

**SEE ALSO**

*crypt*(1), *ed*(1), *grep*(1),  
*terminfo*(4) in the Software Development System manual.

**BUGS**

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options [e.g., *crypt*(1)], terminal settings may not be restored correctly.

## NAME

`pr` - print files

## SYNOPSIS

`pr` [ options ] [ files ]

## DESCRIPTION

`Pr` prints the named files on the standard output. If *file* is `-`, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the `-s` option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until `pr` has completed printing.

The below *options* may appear singly or be combined in any order:

- `+k` Print printing with page *k* (default is 1).
- `-k` Produce *k*-column output (default is 1). The options `-e` and `-i` are assumed for multicolumn output.
- `-a` Print multicolumn output across the page.
- `-m` Merge and print all files simultaneously, one per column (overrides the `-k`, and `-a` options).
- `-d` Double-space the output.
- `-eck` Expand *input* tabs to character positions  $k+1$ ,  $2\cdot k+1$ ,  $3\cdot k+1$ , etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).
- `-ick` In *output*, replace white space wherever possible by inserting tabs to character positions  $k+1$ ,  $2\cdot k+1$ ,  $3\cdot k+1$ , etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *r* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).
- `-nck` Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first  $k+1$  character positions of each column of normal output or each line of `-m` output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).
- `-wk` Set the width of a line to *k* character positions (default is 72 for equal-width multicolumn output, no limit otherwise).
- `-ok` Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.
- `-lk` Set the length of a page to *k* lines (default is 66).
- `-b` Use the next argument as the header to be printed instead of the file name.
- `-p` Pause before beginning each page if the output is directed to a terminal (`pr` will ring the bell at the terminal and wait for a carriage return).

## PR(1)

- f Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.
- r Print no diagnostic reports on failure to open files.
- t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.
- sc Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

### EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, ... :

```
pr -e9 -t <file1 >file2
```

### FILES

**/dev/tty\*** to suspend messages

### SEE ALSO

**cat(1)**.

## NAME

prof - display profile data

## SYNOPSIS

prof [-tcan] [-ox] [-g] [-z] [-h] [-s] [-k] [-m mdata] [prog]

## DESCRIPTION

*Prof* interprets a profile file produced by the *monitor(3C)* function. The symbol table in the object file *prog* (*a.out* by default) is read and correlated with a profile file (*mon.out* by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options *t*, *c*, *a*, and *n* determine the type of sorting of the output lines:

- t Sort by decreasing percentage of total time (default).
- c Sort by decreasing number of calls.
- a Sort by increasing symbol address.
- n Sort lexically by symbol name.

The mutually exclusive options *o* and *x* specify the printing of the address of each symbol monitored:

- o Print each symbol address (in octal) along with the symbol name.
- x Print each symbol address (in hexadecimal) along with the symbol name.

The following options may be used in any combination:

- g Include non-global symbols (static functions).
- k Include information on program stack usage at end of table.
- z Include all symbols in the profile range [see *monitor(3C)*], even if associated with zero number of calls and zero time.
- h Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)
- s Print a summary of several of the monitoring parameters and statistics on the standard error output.
- m mdata

Use file *mdata* instead of *mon.out* as the input profile file.

A program creates a profile file if it has been loaded with the *-p* option of *cc(1)*. This option to the *cc* command arranges for calls to *monitor(3C)* at the beginning and end of execution. It is the call to *monitor* at the end of execution that causes a profile file to be written. The number of calls to a function is tallied if the *-p* option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable *PROFDIR*. If *PROFDIR* does not exist, "mon.out" is produced in the directory current when the program terminates. If *PROFDIR* = string, "string/pid.progname" is produced, where *progname* consists of *argv[0]* with any path prefix removed, and *pid* is the program's process id. If *PROFDIR* = nothing, no profiling output is produced.

A single function may be split into subfunctions for profiling by means of the *MARK* macro [see *prof(5)*].

## PROF(1)

### FILES

mon.out for profile  
a.out for namelist

### SEE ALSO

cc(1),  
exit(2), profil(2), monitor(3C), prof(5) in the Software Development System manual.

### WARNING

The times reported in successive identical runs may show variances of 20% or more, because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may "beat" with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely, however.

### BUGS

Only programs that call *exit(2)* or return from *main* will cause a profile file to be produced, unless a final call to *monitor* is explicitly coded.

The use of the *-p* option *cc(1)* to invoke profiling imposes a limit of 600 (300 on the PDP-11) functions that may have call counters established during program execution. For more counters you must call *monitor(3C)* directly. If this limit is exceeded, other data will be overwritten and the *mon.out* file will be corrupted. The number of call counters used will be reported automatically by the *prof* command whenever the number exceeds 5/6 of the maximum.

## NAME

*prfld*, *prfstat*, *prfdc*, *prfsnap*, *prfpr* — operating system profiler

## SYNOPSIS

```

/etc/prfld [namelist]
/etc/prfstat on
/etc/prfstat off
/etc/prfdc file [period [off_hour]]
/etc/prfsnap file
/etc/prfpr file [cutoff [namelist]]

```

## DESCRIPTION

*Prfld*, *prfstat*, *prfdc*, *prfsnap*, and *prfpr* form a system of programs to facilitate an activity study of the UNIX operating system.

*Prfld* is used to initialize the recording mechanism in the system. It generates a table containing the starting address of each system subroutine as extracted from *namelist*.

*Prfstat* is used to enable or disable the sampling mechanism. Profiler overhead is less than 1% as calculated for 500 text addresses. *Prfstat* will also reveal the number of text addresses being measured.

*Prfdc* and *prfsnap* perform the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed. *Prfdc* will store the counters into *file* every *period* minutes and will turn off at *off\_hour* (valid values for *off\_hour* are 0-24). *Prfsnap* collects data at the time of invocation only, appending the counter values to *file*.

*Prfpr* formats the data collected by *prfdc* or *prfsnap*. Each text address is converted to the nearest text symbol (as found in *namelist*) and is printed if the percent activity for that range is greater than *cutoff*.

## FILES

```

/dev/prf interface to profile data and text addresses
/unix default for namelist file

```

## SEE ALSO

*prf*(7).

## PRS(1)

### NAME

*prs* - print an SCCS file

### SYNOPSIS

*prs* [-d[*dataspec*]] [-r[SID]] [-e] [-l] [-c[*date-time*]] [-a] files

### DESCRIPTION

*Prs* prints, on the standard output, parts or all of an SCCS file [see *sccsfile(4)*] in a user-supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.), and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to *prs*, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

- d[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.
- r[SID] Used to specify the SCCS IDentification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.
- e Requests information for all deltas created *earlier* than and including the delta designated via the -r keyletter or the date given by the -c option.
- l Requests information for all deltas created *later* than and including the delta designated via the -r keyletter or the date given by the -c option.
- c[*date-time*] The cutoff date-time -c[cutoff] is in the form:

YY[MM[DD[HH[MM[SS]]]]]

Units omitted from the date-time default to their maximum possible values; that is, -c7502 is equivalent to -c750228235959. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date in the form: "-c77/2/2 9:22:25".

- a Requests printing of information for both removed, i.e., delta type = R, [see *rmDEL(1)*] and existing, i.e., delta type = D, deltas. If the -a keyletter is not specified, information for existing deltas only is provided.

### DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file [see *sccsfile(4)*] have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of



a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords. A tab is specified by \t and carriage return/new-line is specified by \n. The default data keywords are:

":Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"

TABLE 1. SCCS Files Data Keywords

| Keyword | Data Item                               | File Section | Value         | Format |
|---------|-----------------------------------------|--------------|---------------|--------|
| :Dt:    | Delta information                       | Delta Table  | See below*    | S      |
| :DL:    | Delta line statistics                   | "            | :Li/:Ld/:Lu:  | S      |
| :Li:    | Lines inserted by Delta                 | "            | nnnnn         | S      |
| :Ld:    | Lines deleted by Delta                  | "            | nnnnn         | S      |
| :Lu:    | Lines unchanged by Delta                | "            | nnnnn         | S      |
| :DT:    | Delta type                              | "            | D or R        | S      |
| :I:     | SCCS ID string (SID)                    | "            | :R::L::B::S:  | S      |
| :R:     | Release number                          | "            | nnnn          | S      |
| :L:     | Level number                            | "            | nnnn          | S      |
| :B:     | Branch number                           | "            | nnnn          | S      |
| :S:     | Sequence number                         | "            | nnnn          | S      |
| :D:     | Date Delta created                      | "            | :Dy/:Dm/:Dd:  | S      |
| :Dy:    | Year Delta created                      | "            | nn            | S      |
| :Dm:    | Month Delta created                     | "            | nn            | S      |
| :Dd:    | Day Delta created                       | "            | nn            | S      |
| :T:     | Time Delta created                      | "            | :Th::Tm::Ts:  | S      |
| :Th:    | Hour Delta created                      | "            | nn            | S      |
| :Tm:    | Minutes Delta created                   | "            | nn            | S      |
| :Ts:    | Seconds Delta created                   | "            | nn            | S      |
| :P:     | Programmer who created Delta            | "            | logname       | S      |
| :DS:    | Delta sequence number                   | "            | nnnn          | S      |
| :DP:    | Predecessor Delta seq-no.               | "            | nnnn          | S      |
| :DI:    | Seq-no. of deltas incl., excl., ignored | "            | :Dn/:Dx/:Dg:  | S      |
| :Dn:    | Deltas included (seq #)                 | "            | :DS: :DS: ... | S      |
| :Dx:    | Deltas excluded (seq #)                 | "            | :DS: :DS: ... | S      |
| :Dg:    | Deltas ignored (seq #)                  | "            | :DS: :DS: ... | S      |
| :MR:    | MR numbers for delta                    | "            | text          | M      |
| :C:     | Comments for delta                      | "            | text          | M      |
| :UN:    | User names                              | User Names   | text          | M      |
| :FL:    | Flag list                               | Flags        | text          | M      |
| :Y:     | Module type flag                        | "            | text          | S      |
| :MF:    | MR validation flag                      | "            | yes or no     | S      |
| :MP:    | MR validation pgm name                  | "            | text          | S      |
| :KF:    | Keyword error/warning flag              | "            | yes or no     | S      |
| :KV:    | Keyword validation string               | "            | text          | S      |
| :BF:    | Branch flag                             | "            | yes or no     | S      |
| :J:     | Joint edit flag                         | "            | yes or no     | S      |
| :LK:    | Locked releases                         | "            | :R: ...       | S      |
| :Q:     | User-defined keyword                    | "            | text          | S      |
| :M:     | Module name                             | "            | text          | S      |
| :FB:    | Floor boundary                          | "            | :R:           | S      |
| :CB:    | Ceiling boundary                        | "            | :R:           | S      |
| :Ds:    | Default SID                             | "            | :I:           | S      |
| :ND:    | Null delta flag                         | "            | yes or no     | S      |
| :FD:    | File descriptive text                   | Comments     | text          | M      |

## PRS(1)

|                                         |                                  |      |                  |   |
|-----------------------------------------|----------------------------------|------|------------------|---|
| :BD:                                    | Body                             | Body | text             | M |
| :GB:                                    | Getten body                      | "    | text             | M |
| :W:                                     | A form of <i>what</i> (1) string | N/A  | :Z::M:\t:l:      | S |
| :A:                                     | A form of <i>what</i> (1) string | N/A  | :Z::Y: :M: l::Z: | S |
| :Z:                                     | <i>what</i> (1) string delimiter | N/A  | @(#)             | S |
| :F:                                     | SCCS file name                   | N/A  | text             | S |
| :PN:                                    | SCCS file path name              | N/A  | text             | S |
| * :Dt: = :DT: :l: :D: :T: :P: :DS: :DP: |                                  |      |                  |   |

### EXAMPLES

```
prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file
may produce on the standard output:
```

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

```
prs -d"Newest delta for pgm :M:: :l: Created :D: By :P:" -r s.file
may produce on the standard output:
```

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

As a *special case*:

```
prs s.file
may produce on the standard output:
D 1.1 77/12/1 00:00:00 cas 1 000000/000000/000000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the *-a* keyletter.

### FILES

```
/tmp/pr????
```

### SEE ALSO

admin(1), delta(1), get(1), help(1).  
sccsfile(4) and "Source Code Control System User Guide" in the Software Development System manual.

### DIAGNOSTICS

Use *help*(1) for explanations.

**NAME**

**ps** — report process status

**SYNOPSIS**

**ps** [ options ]

**DESCRIPTION**

*Ps* prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

*Options* using lists as arguments can have the list specified in one of two forms: a list of identifiers separated from one another by a comma, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

The *options* are:

- e Print information about all processes.
- d Print information about all processes, except process group leaders.
- a Print information about all processes, except process group leaders and processes not associated with a terminal.
- f Generate a *full* listing. (See below for meaning of columns in a full listing).
- l Generate a *long* listing. See below.
- c *corefile* Use the file *corefile* in place of */dev/mem*.
- s *swapdev* Use the file *swapdev* in place of */dev/swap*. This is useful when examining a *corefile*; a *swapdev* of */dev/null* will cause the user block to be zeroed out.
- n *namelist* The argument will be taken as the name of an alternate system *namelist* file in place of */unix*.
- t *termlist* Restrict listing to data about the processes associated with the terminals given in *termlist*. Terminal identifiers may be specified in one of two forms: the device's file name (e.g., *tty04*) or if the device's file name starts with *tty*, just the digit identifier (e.g., *04*).
- p *proclist* Restrict listing to data about processes whose process ID numbers are given in *proclist*.
- u *uidlist* Restrict listing to data about processes whose user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID will be printed unless the *-f* option is used, in which case the login name will be printed.
- g *grplist* Restrict listing to data about processes whose process group leaders are given in *grplist*.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters *f* and *l* indicate the option (*full* or *long*) that causes the corresponding heading to appear; *all* means that the heading always appears. Note that these two options determine only what information is provided for a process; they do *not* determine which processes will be listed.

- F** (i) Flags (octal and additive) associated with the process:
- |    |                                          |
|----|------------------------------------------|
| 0  | swapped;                                 |
| 1  | in core;                                 |
| 2  | system process;                          |
| 4  | locked-in core (e.g., for physical I/O); |
| 10 | being swapped;                           |

|              |       |                                                                                                                                                         |
|--------------|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | 20    | being traced by another process;                                                                                                                        |
|              | 40    | another tracing flag;                                                                                                                                   |
|              | 100   | 3B20 computer: swapin segment expansion;<br>VAX-11/780 and iAPX 286: text pointer valid;                                                                |
|              | 200   | 3B20 computer: process is child (during fork swap);<br>VAX-11/780: process is partially swapped.                                                        |
| <b>S</b>     | (i)   | The state of the process:<br>0 non-existent;<br>S sleeping;<br>W waiting;<br>R running;<br>I intermediate;<br>Z terminated;<br>T stopped;<br>X growing. |
| <b>UID</b>   | (f,i) | The user ID number of the process owner; the login name is printed under the <b>-f</b> option.                                                          |
| <b>PID</b>   | (all) | The process ID of the process; it is possible to kill a process if you know this datum.                                                                 |
| <b>PPID</b>  | (f,i) | The process ID of the parent process.                                                                                                                   |
| <b>C</b>     | (f,i) | Processor utilization for scheduling.                                                                                                                   |
| <b>PRI</b>   | (i)   | The priority of the process; higher numbers mean lower priority.                                                                                        |
| <b>NI</b>    | (i)   | Nice value; used in priority computation.                                                                                                               |
| <b>ADDR</b>  | (i)   | The memory address of the process (a pointer to the segment table array on the 3B20 computer), if resident; otherwise, the disk address.                |
| <b>SZ</b>    | (i)   | The size in blocks of the core image of the process.                                                                                                    |
| <b>WCHAN</b> | (i)   | The event for which the process is waiting or sleeping; if blank, the process is running.                                                               |
| <b>STIME</b> | (f)   | Starting time of the process.                                                                                                                           |
| <b>TTY</b>   | (all) | The controlling terminal for the process.                                                                                                               |
| <b>TIME</b>  | (all) | The cumulative execution time for the process.                                                                                                          |
| <b>CMD</b>   | (all) | The command name; the full command name and its arguments are printed under the <b>-f</b> option.                                                       |

A process that has **exited** and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Under the **-f** option, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the **-f** option, is printed in square brackets.

## FILES

|                     |                                         |
|---------------------|-----------------------------------------|
| <b>/unix</b>        | system name list                        |
| <b>/dev/mem</b>     | memory                                  |
| <b>/dev/swap</b>    | the default swap device                 |
| <b>/etc/passwd</b>  | supplies UID information                |
| <b>/etc/ps_data</b> | internal data structure                 |
| <b>/dev</b>         | searched to find terminal ("tty") names |

## SEE ALSO

acctcom(1), kill(1), nice(1).

## BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

**NAME**

`ptx` — permuted index

**SYNOPSIS**

`ptx` [ options ] [ input [ output ] ]

**DESCRIPTION**

`Ptx` generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of each line. `Ptx` output is in the form:

```
.xx "tail" "before keyword" "keyword and after" "head"
```

where `.xx` is assumed to be an `nroff` or `troff`(1) macro provided by the user, or provided by the `mptx`(5) macro package. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The following *options* can be applied:

- `-f`        Fold upper and lowercase letters for sorting.
- `-t`        Prepare the output for the phototypesetter.
- `-w n`      Use the next argument, *n*, as the length of the output line. The default line length is 72 characters for `nroff` and 100 for `troff`.
- `-g n`      Use the next argument, *n*, as the number of characters that `ptx` will reserve in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3.
- `-o only`    Use as keywords only the words given in the *only* file.
- `-i ignore`   Do not use as keywords any words given in the *ignore* file. If the `-i` and `-o` options are missing, use `/usr/lib/eign` as the *ignore* file.
- `-b break`   Use the characters in the *break* file to separate words. Tab, newline, and space characters are *always* used as break characters.
- `-r`        Take any leading non-blank characters of each input line to be a reference identifier (as to a page or chapter), separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using `ptx`.

**FILES**

```
/bin/sort
/usr/lib/eign
/usr/lib/tmac/tmac.ptx
```

**SEE ALSO**

`mm`(5), `mptx`(5) in the Software Development System manual.  
 "Nroff and Troff User Manual" in the Text Preparation System manual.

**BUGS**

Line length counts do not account for overstriking or proportional spacing. Lines that contain tildes (~) are botched, because `ptx` uses that character internally.

**NAME**

pwck, grpck — password/group file checkers

**SYNOPSIS**

*/etc/pwck* [file]  
*/etc/grpck* [file]

**DESCRIPTION**

*Pwck* scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The criteria for determining a valid login name is derived from *Setting up the UNIX System in the UNIX System V Administrator Guide*. The default password file is */etc/passwd*.

*Grpck* verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is */etc/group*.

**FILES**

*/etc/group*  
*/etc/passwd*

**SEE ALSO**

*group(4)*, *passwd(4)* in the Software Development System manual.

**DIAGNOSTICS**

Group entries in */etc/group* with no login names are flagged.

**NAME**

`pwd` -- working directory name

**SYNOPSIS**

`pwd`

**DESCRIPTION**

*Pwd* prints the path name of the working (current) directory.

**SEE ALSO**

`cd(1)`.

**DIAGNOSTICS**

"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred to a UNIX system programming counselor.

# RATFOR(1)

## NAME

ratfor — rational Fortran dialect

## SYNOPSIS

**ratfor** [ options ] [ files ]

## DESCRIPTION

*Ratfor* converts a rational dialect of Fortran into ordinary irrational Fortran.

*Ratfor* provides control flow constructs essentially identical to those in C:

```
statement grouping:
 { statement; statement; statement }

decision-making:
 if (condition) statement [else statement]
 switch (integer value) {
 case integer: statement
 ...
 [default:] statement
 }
```

```
loops:
 while (condition) statement
 for (expression; condition; expression) statement
 do limits statement
 repeat statement [until (condition)]
 break
 next
```

and some syntactic sugar to make programs easier to read and write:

```
free form input:
 multiple statements/line; automatic continuation
```

```
comments:
 # this is a comment.
```

```
translation of relationals:
 >, >=, etc., become .GT., .GE., etc.
```

```
return expression to caller from function:
 return (expression)
```

```
define:
 define name replacement
```

```
include:
 include file
```

The option **-h** causes quoted strings to be turned into **27H** constructs. The **-C** option copies comments to the output and attempts to format it neatly. Normally, continuation lines are marked with a **&** in column 1; the option **-6x** makes the continuation character **x** and places it in column 6.

*Ratfor* is best used with *f77(1)*.

## SEE ALSO

*efl(1)*, *f77(1)*.

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.



## NAME

regcmp — regular expression compile

## SYNOPSIS

regcmp [ - ] files

## DESCRIPTION

*Regcmp*, in most cases, precludes the need for calling *regcmp*(3X) from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file.i*. If the *-* option is used, the output will be placed in *file.c*. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *File.i* files may thus be *included* into C programs, or *file.c* files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex(abc,line)* will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

## EXAMPLES

```
name "([A-Za-z][A-Za-z0-9_]+)$0"
telno "\((0,1)([2-9][01][1-9])$0\)(0,1) *"
 "([2-9][0-9]{2})$1[-](0,1)"
 "([0-9]{4})$2"
```

In the C program that uses the *regcmp* output,  
 regex(telno, line, area, exch, rest)

will apply the regular expression named *telno* to *line*.

## SEE ALSO

regcmp(3X) in the Software Development System manual.

## RJESTAT(1C)

### NAME

*rjestat* — RJE status report and interactive status console

### SYNOPSIS

*rjestat* [ *host* ]... [ *-shost* ] [ *-chost cmd* ] [ *-jhost jobname* ]..

### DESCRIPTION

*Rjestat* provides a method of determining the status of an RJE link and of simulating an IBM remote console (with UNIX system features added). When invoked with no arguments, *rjestat* reports the current status of all the RJE links connected to the UNIX system. The options are:

*host*           Print the status of the line to *host*. *Host* is the pseudonym for a particular IBM system. It can be any name that corresponds to one in the first column of the RJE configuration file.

*-shost*        After all the arguments have been processed, start an interactive status console to *host*.

*-chost cmd*   Interpret *cmd* as if it were entered in status console mode to *host*. See below for the proper format of *cmd*.

*-jhost jobname*  
Print all status pertaining to a user job with name *jobname* that has been sent by the *host* system to the *rje* system.

In status console mode, *rjestat* prompts with the host pseudonym followed by : whenever it is ready to accept a command. Commands are terminated with a new-line. A line that begins with ! is sent to the UNIX system shell for execution. A line that begins with the letter q terminates *rjestat*. All other input lines are assumed to have the form:

*ibmcmd* [ *redirect* ]

*Ibmcmd* is any IBM JES or HASP command. Only the super-user or *rje* login can send commands other than display or inquiry commands. *Redirect* is a pipeline or a redirection to a file (e.g., "> file" or " | grep ..."). The IBM response is written to the pipeline or file. If *redirect* is not present, the response is written to the standard output of *rjestat*.

An interrupt signal (DEL or BREAK) will cancel the command in progress and cause *rjestat* to return to the command input mode.

### EXAMPLE

The following command reports the status of all the card readers attached to host A, remote 5. JES2 is assumed.

```
rjestat -cA '$du,rmt5 | grep RD'
```

### DIAGNOSTICS

The message "RJE error: ..." indicates that *rjestat* found an inconsistency in the RJE system. This may be transient but should be reported to the site administrator.

### FILES

*/usr/rje/lines* RJE configuration file  
*resp* host response file that exists in the RJE subsystem directory (e.g., */usr/rje1*).

### SEE ALSO

*send(1C)*.

**NAME**

`rm, rmdir` - remove files or directories

**SYNOPSIS**

`rm [ -fri ] file ...`

`rmdir dir ...`

**DESCRIPTION**

*Rm* removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with `y` the file is deleted, otherwise the file remains. No questions are asked when the `-f` option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument `-r` has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the `-i` (interactive) option is in effect, *rm* asks whether to delete each file, and, under `-r`, whether to examine each directory.

*Rmdir* removes entries for the named directories, which must be empty.

**SEE ALSO**

`unlink(2)` in the Software Development System manual.

**DIAGNOSTICS**

Generally self-explanatory. It is forbidden to remove the file `..` merely to avoid the antisocial consequences of inadvertently doing something like:

```
rm -r .*
```

## RMDEL(1)

### NAME

`rmdel` - remove a delta from an SCCS file

### SYNOPSIS

`rmdel -rSID files`

### DESCRIPTION

*Rmdel* removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* (see *get(1)*) exists for the named SCCS file, the specified must *not* appear in any entry of the *p-file*).

If a directory is named, *rmdel* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with *s.*) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User Guide*. Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

### FILES

x.file        (see *delta(1)*)  
z.file        (see *delta(1)*)

### SEE ALSO

*delta(1)*, *get(1)*, *help(1)*, *prc(1)*,  
*sccsfile(4)* and "Source Code Control System User Guide" in the Software Development System manual.

### DIAGNOSTICS

Use *help(1)* for explanations.

**NAME**

*sact* - print current SCCS file editing activity

**SYNOPSIS**

*sact* files

**DESCRIPTION**

*Sact* informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the *-e* option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of *-* is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

- |         |                                                                                                                          |
|---------|--------------------------------------------------------------------------------------------------------------------------|
| Field 1 | specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta. |
| Field 2 | specifies the SID for the new delta to be created.                                                                       |
| Field 3 | contains the logname of the user who will make the delta (i.e., executed a <i>get</i> for editing).                      |
| Field 4 | contains the date that <i>get -e</i> was executed.                                                                       |
| Field 5 | contains the time that <i>get -e</i> was executed.                                                                       |

**SEE ALSO**

*delta*(1), *get*(1), *unget*(1).

**DIAGNOSTICS**

Use *help*(1) for explanations.

## SADP(1M)

### NAME

sadp — disk access profiler

### SYNOPSIS

**sadp** [ **-t** ] [ **-d** device[**-ctrnumber**] ] s [ n ]

### DESCRIPTION

*Sadp* reports disk access location and seek distance, in tabular or histogram form. It samples disk activity once every second during an interval of *s* seconds. This is done repeatedly if *n* is specified. Cylinder usage and disk distance are recorded in units of 8 cylinders.

The only valid value of *device* is *wn*. *Ctrnumber* specifies the disk controller. The **-d** option may be omitted, if only one *device* is present.

The **-t** flag causes the data to be reported in tabular form. The **-h** flag produces a histogram on the printer of the data. Default is **-t**.

### EXAMPLE

The command:

```
sadp -d wn-0 900 4
```

will generate 4 tabular reports, each describing cylinder usage and seek distance of Winchester disk controller 0 during a 15-minute interval.

### FILES

/dev/kmem

## NAME

sa1, sa2, sadc - system activity report package

## SYNOPSIS

```
/usr/lib/sa/sadc [t n] [ofile]
```

```
/usr/lib/sa/sa1 [t n]
```

```
/usr/lib/sa/sa2 [-ubdycwaqvmA] [-s time] [-e time] [-i sec]
```

## DESCRIPTION

System activity data can be accessed at the special request of a user [see *sar(1)*] and automatically on a routine basis as described here. The operating system contains a number of counters that are incremented as various system actions occur. These include CPU utilization counters, buffer usage counters, disk I/O activity counters, TTY device activity counters, switching and system-call counters, file-access counters, queue activity counters, and counters for interprocess communications.

*Sadc* and shell procedures, *sa1* and *sa2*, are used to sample, save, and process this data.

*Sadc*, the data collector, samples system data *n* times every *t* seconds and writes in binary format to *ofile* or to standard output. If *t* and *n* are omitted, a special record is written. This facility is used at system boot time to mark the time at which the counters restart from zero. The */etc/rc* entry:

```
su sys -c "/usr/lib/sa/sadc /usr/adm/sa/sadate +%d"
```

writes the special record to the daily data file to mark the system restart.

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file */usr/adm/sa/sadd* where *dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted. The entries in **crontab** [see *cron(1M)*]:

```
0 * * * 0,6 su sys -c "/usr/lib/sa/sa1"
0 8-17 * * 1-5 su sys -c "/usr/lib/sa/sa1 1200 3"
0 18-7 * * 1-5 su sys -c "/usr/lib/sa/sa1"
```

will produce records every 20 minutes during working hours and hourly otherwise.

The shell script *sa2*, a variant of *sar(1)*, writes a daily report in file */usr/adm/sa/sardd*. The options are explained in *sar(1)*. The **crontab** entry:

```
5 18 * * 1-5 su adm -c "/usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600
-A"
```

will report important activities hourly during the working day.

## SAR(1M)

The structure of the binary daily data file is:

```
struct sa {
 struct sysinfo si; /* see /usr/include/sys/sysinfo.h */
 int szinode; /* current entries of i-node table */
 int szfile; /* current entries of file table */
 int sztext; /* current entries of text table */
 int szproc; /* current entries of proc table */
 int szlckf; /* current size of file record header table */
 int szlckr; /* current size of file record lock table */
 int mszinode; /* size of i-node table */
 int mszfile; /* size of file table */
 int msztext; /* size of text table */
 int mszproc; /* size of proc table */
 int mszlckf; /* maximum size of file record header table */
 int mszlckr; /* maximum size of file record lock table */
 long inodeovf; /* cumul. overflows of i-node table */
 long fileovf; /* cumul. overflows of file table */
 long textovf; /* cumul. overflows of text table */
 long procovf; /* cumul. overflows of proc table */
 time_t ts; /* time stamp, seconds */
 long devio[NDEVS][4]; /* device info for up to NDEVS units */
#define IO_OPS 0 /* cumul. I/O requests */
#define IO_BCNT 1 /* cumul. blocks transferred */
#define IO_ACT 2 /* cumul. drive busy time in ticks */
#define IO_RESP 3 /* cumul. I/O resp time in ticks */
};
```

### FILES

|                   |                   |
|-------------------|-------------------|
| /usr/adm/sa/sadd  | daily data file   |
| /usr/adm/sa/sardd | daily report file |
| /tmp/sa.adrfl     | address file      |

### SEE ALSO

cron(1M), sag(1G), sar(1), timex(1).



**NAME**

sa1, sa2, sadc — system activity report package

**SYNOPSIS**

```
/usr/lib/sa/sadc [t n] [ofile]
/usr/lib/sa/sa1 [t n]
/usr/lib/sa/sa2 [-ubdycwaqvmA] [-s time] [-e time] [-i sec]
```

**DESCRIPTION**

System activity data can be accessed at the special request of a user [see *sar(1)*] and automatically on a routine basis as described here. The operating system contains a number of counters that are incremented as various system actions occur. These include CPU utilization counters, buffer usage counters, disk I/O activity counters, TTY device activity counters, switching and system-call counters, file-access counters, queue activity counters, and counters for interprocess communications.

*Sadc* and shell procedures, *sa1* and *sa2*, are used to sample, save, and process this data.

*Sadc*, the data collector, samples system data *n* times every *t* seconds and writes in binary format to *ofile* or to standard output. If *t* and *n* are omitted, a special record is written. This facility is used at system boot time to mark the time at which the counters restart from zero. The */etc/rc* entry:

```
su sys -c "/usr/lib/sa/sadc /usr/adm/sa/sadate +%d"
```

writes the special record to the daily data file to mark the system restart.

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file */usr/adm/sa/sadd* where *dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted. The entries in *crontab* [see *cron(1M)*]:

```
0 * * * 0,6 su sys -c "/usr/lib/sa/sa1"
0 8-17 * * 1-5 su sys -c "/usr/lib/sa/sa1 1200 3"
0 18-7 * * 1-5 su sys -c "/usr/lib/sa/sa1"
```

will produce records every 20 minutes during working hours and hourly otherwise.

The shell script *sa2*, a variant of *sar(1)*, writes a daily report in file */usr/adm/sa/sar<sub>dd</sub>*. The options are explained in *sar(1)*. The *crontab* entry:

```
5 18 * * 1-5 su adm -c "/usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600
-A"
```

will report important activities hourly during the working day.

## SAR(1M)

The structure of the binary daily data file is:

```
struct sa {
 struct sysinfo si; /* see /usr/include/sys/sysinfo.h */
 int szi-node; /* current entries of i-node table */
 int szfile; /* current entries of file table */
 int sztext; /* current entries of text table */
 int szproc; /* current entries of proc table */
 int mszi-node; /* size of i-node table */
 int mszfile; /* size of file table */
 int msztext; /* size of text table */
 int mszproc; /* size of proc table */
 long i-nodeovf; /* cumul. overflows of i-node table */
 long fileovf; /* cumul. overflows of file table */
 long textovf; /* cumul. overflows of text table */
 long procovf; /* cumul. overflows of proc table */
 time_t ts; /* time stamp, seconds */
 long devio[NDEVS][4]; /* device info for up to NDEVS units */
#define IO_OPS 0 /* cumul. I/O requests */
#define IO_BCNT 1 /* cumul. blocks transferred */
#define IO_ACT 2 /* cumul. drive busy time in ticks */
#define IO_RESP 3 /* cumul. I/O resp time in ticks */
};
```

### FILES

```
/usr/adm/sa/sadd daily data file
/usr/adm/sa/saradd daily report file
/tmp/sa.adrfl address file
```

### SEE ALSO

cron(1M), sag(1G), sar(1), timex(1).

**NAME**

`sccsdiff` — compare two versions of an SCCS file

**SYNOPSIS**

`sccsdiff -rSID1 -rSID2 [-p] [-sn] files`

**DESCRIPTION**

`Sccsdiff` compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

- `-rSID?` *SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to `bdiff(1)` in the order given.
- `-p` pipe output for each file through `pr(1)`.
- `-sn` *n* is the file segment size that `bdiff` will pass to `diff(1)`. This is useful when `diff` fails due to a high system load.

**FILES**

`/tmp/get????` Temporary files

**SEE ALSO**

`bdiff(1)`, `get(1)`, `help(1)`, `pr(1)`.

"Source Code Control System User Guide" in the Software Development System manual.

**DIAGNOSTICS**

"*file*: No differences" If the two versions are the same.  
Use `help(1)` for explanations.

## SDB(1)

### NAME

sdb - symbolic debugger

### SYNOPSIS

sdb [-w] [-W] [ objfil [ corfil [ directory-list ] ] ]

### DESCRIPTION

*Sdb* is a symbolic debugger that can be used with C and F77 programs. It may be used to examine their object files and core files and to provide a controlled environment for their execution.

*Objfil* is normally an executable program file which has been compiled with the `-g` (debug) option; if it has not been compiled with the `-g` option, or if it is not an executable file, the symbolic capabilities of *sdb* will be limited, but the file can still be examined and the program debugged. The default for *objfil* is `a.out`. *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is `core`. The core file need not be present. A `-` in place of *corfil* will force *sdb* to ignore any core image file. The colon separated list of directories (*directory-list*) is used to locate the source files used to build *objfil*.

It is useful to know that at any time there is a *current line* and *current file*. If *corfil* exists then they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in *main()*. The current line and file may be changed with the source file examination commands.

By default, warnings are provided if the source files used in producing *objfil* cannot be found, or are newer than *objfil*. This checking feature and the accompanying warnings may be disabled by the use of the `-W` flag.

Names of variables are written just as they are in C or F77. Note that names in C are now of arbitrary length, *sdb* will no longer truncate names. Variables local to a procedure may be accessed using the form *procedure:variable*. If no procedure name is given, the procedure containing the current line is used by default.

It is also possible to refer to structure members as *variable.member*, pointers to structure members as *variable->member* and array elements as *variable[number]*. Pointers may be dereferenced by using the form *pointer[0]*. Combinations of these forms may also be used. F77 common variables may be referenced by using the name of the common block instead of the structure name. Blank common variables may be named by the form *.variable*. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by *sdb*. An unqualified structure variable may also be used with various commands. Generally, *sdb* will interpret a structure as a set of variables. Thus, *sdb* will display the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value *sdb* displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as *variable[number][number]...*, or as *variable[number,number,...]*. In place of *number*, the form *number;number* may be used to indicate a range of values, `*` may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, *sdb* displays all the values of an array or of the section of an array if trailing subscripts are omitted. It displays only the address of the array itself or of the section specified by the user if subscripts

are omitted. A multidimensional parameter in an F77 program cannot be displayed as an array, but it is actually a pointer, whose value is the location of the array. The array itself can be accessed symbolically from the calling function.

A particular instance of a variable on the stack may be referenced by using the form *procedure:variable,number*. All the variations mentioned in naming variables may be used. *Number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C may be used, so that addresses may be input in decimal, octal or hexadecimal.

Line numbers in the source program are referred to as *file-name:number* or *procedure:number*. In either case the number is relative to the beginning of the file. If no procedure or file name is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under *sdb*, all addresses refer to the executing program; otherwise they refer to *objfil* or *corfil*. An initial argument of *-w* permits overwriting locations in *objfil*.

#### Addresses

The address in a file associated with a written address is determined by a mapping associated with that file. For the iAPX286, which has areas of up to 64k, the mapping is performed effectively by an array of triples. These triples are interpreted as follows:

|             |                              |
|-------------|------------------------------|
| <b>bil</b>  | = address (32 bits)          |
| <b>elil</b> | = size of area               |
| <b>fil</b>  | = file address for this area |

For *a.out* and *core* files, the address will be interpreted as an iAPX286 selector/displacement, where the top 13 bits are used as an index into the array of triples. Normally, the user will not have to form this address as it will be provided by the SDB mechanism. The privilege and system bit part of the address will be ignored, which will cause an error if they are incorrectly specified in a running system.

For files other than *a.out* and *core* files, where the user wishes to look at the file with no address translation, the system will set up an array of 64k-byte triples to cover the maximum file size. The last entry will contain the residue if the maximum file size is not a multiple of 64k bytes. When you specify addresses for these files, the address must be linear. The top 16 bits of the address will be used to index the array of triples.

In order for *sdb* to be used on large files, all appropriate values are kept as signed 32-bit integers.

#### Commands

The commands for examining data in the program are:

- t** Print a stack trace of the terminated or halted program.
- T** Print the top line of the stack trace.

#### *variable/clm*

Print the value of *variable* according to length *l* and format *m*. A numeric count *c* indicates that a region of memory, beginning at the address implied by *variable*, is to be displayed. The length specifiers are:

|          |          |
|----------|----------|
| <b>b</b> | one byte |
|----------|----------|

**b** two bytes (half word)  
**l** four bytes (long word)

Legal values for *m* are:

**c** character  
**d** decimal  
**u** decimal, unsigned  
**o** octal  
**x** hexadecimal  
**f** 32-bit single precision floating point  
**g** 64-bit double precision floating point  
**s** Assume *variable* is a string pointer and print characters starting at the address pointed to by the variable.  
**a** Print characters starting at the variable's address. This format may not be used with register variables.  
**p** pointer to procedure  
**i** disassemble machine-language instruction with addresses printed numerically and symbolically.  
**I** disassemble machine-language instruction with addresses just printed numerically.

The length specifiers are only effective with the formats **c**, **d**, **u**, **o** and **x**. Any of the specifiers, *c*, *l*, and *m*, may be omitted. If all are omitted, *sdb* chooses a length and a format suitable for the variable's type as declared in the program. If *m* is specified, then this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier *c* tells *sdb* to display that many units of memory, beginning at the address of *variable*. The number of bytes in one such unit of memory is determined by the length specifier *l*, or if no length is given, by the size associated with the *variable*. If a count specifier is used for the **s** or a command, then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command *./*.

The *sh*(1) metacharacters **\*** and **?** may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, variables local to the current procedure and global variables are matched; if a procedure name is specified then only variables local to that procedure are matched. To match only global variables, the form *:pattern* is used.

*linenumber?**lm*

*variable?**lm*

Print the value at the address from **a**, **out** or **I** space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is *"i"*.

*variable =lm*

*linenumber =lm*

*number =lm*

Print the address of *variable* or *linenumber*, or the value of *number*, in the format specified by *lm*. If no format is given, then **lx** is used. The last variant of this command provides a convenient way to convert between decimal, octal and hexadecimal.

*variable!value*

Set *variable* to the given *value*. The value may be a number, a character constant or a variable. The value must be well defined; expressions which produce more than one value, such as structures, are not allowed. Character constants are denoted *'character'*. Numbers are viewed as integers

as having the type double. Registers are viewed as integers. The *variable* may be an expression which indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type *int*. C conventions are used in any type conversions necessary to perform the indicated assignment.

- x Print the machine registers and the current machine-language instruction.
- X Print the current machine-language instruction.

The commands for examining source files are:

*e procedure*  
*e file-name*  
*e directory/*  
*e directory file-name*

The first two forms set the current file to the file containing *procedure* or to *file-name*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The default is the current working directory. The latter two forms change the value of *directory*. If no procedure, file name, or directory is given, the current procedure name and file name are reported.

*/regular expression/*

Search forward from the current line for a line containing a string matching *regular expression* as in *ed(1)*. The trailing */* may be deleted.

*?regular expression?*

Search backward from the current line for a line containing a string matching *regular expression* as in *ed(1)*. The trailing *?* may be deleted.

**p** Print the current line.

**z** Print the current line followed by the next 9 lines. Set the current line to the last line printed.

**w** Window. Print the 10 lines around the current line.

*number*

Set the current line to the given line number. Print the new current line.

*count +*

Advance the current line by *count* lines. Print the new current line.

*count -*

Retreat the current line by *count* lines. Print the new current line.

The commands for controlling the execution of the source program are:

*count r args*

*count R*

Run the program with the given arguments. The *r* command with no arguments reuses the previous arguments to the program while the *R* command runs the program with no arguments. An argument beginning with *<* or *>* causes redirection for the standard input or output, respectively. If *count* is given, it specifies the number of breakpoints to be ignored.

*linenumber c count*

*linenumber C count*

Continue after a breakpoint or interrupt. If *count* is given, it specifies the breakpoint at which to stop after ignoring *count* - 1 breakpoints. *C* continues with the signal which caused the program to stop reactivated and *c* ignores it. If a line number is specified then a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted

when the command finishes.

*linenumber g count*

Continue after a breakpoint with execution resumed at the given line. If *count* is given, it specifies the number of breakpoints to be ignored.

*s count*

**S** *count*

Single step the program through *count* lines. If no *count* is given then the program is run for one line. **S** is equivalent to *s* except it steps through procedure calls.

*i*

**I** Single step by one machine-language instruction. **I** steps with the signal which caused the program to stop reactivated and *i* ignores it.

*variable\$m count*

*address:m count*

Single step (as with *s*) until the specified location is modified with a new value. If *count* is omitted, it is effectively infinity. *Variable* must be accessible from the current procedure. Since this command is done by software, it can be very slow.

*level v*

Toggle verbose mode, for use when single stepping with **S**, *s* or *m*. If *level* is omitted, then just the current source file and/or subroutine name is printed when either changes. If *level* is 1 or greater, each **C** source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A *v* turns verbose mode off if it is on for any level.

**k** Kill the program being debugged.

*procedure(arg1,arg2,...)*

*procedure(arg1,arg2,...)/m*

Execute the named procedure with the given arguments. Arguments can be integer, character or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format *m*. If no format is given, it defaults to *d*.

*linenumber b commands*

Set a breakpoint at the given line. If a procedure name without a line number is given (e.g., "proc:"), a breakpoint is placed at the first line in the procedure even if it was not compiled with the *-g* option. If no *linenumber* is given, a breakpoint is placed at the current line. If no *commands* are given, execution stops just before the breakpoint and control is returned to *sdb*. Otherwise the *commands* are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons. If **k** is used as a command to execute at a breakpoint, control returns to *sdb*, instead of continuing execution.

**B** Print a list of the currently active breakpoints.

*linenumber d*

Delete a breakpoint at the given line. If no *linenumber* is given then the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from the standard input. If the line begins with a *y* or *d* then the breakpoint is deleted.

**D** Delete all breakpoints.



**l** Print the last executed line.

**linenumber a**

Announce. If *linenumber* is of the form *proc:number*, the command effectively does a *linenumber b l*. If *linenumber* is of the form *proc*, the command effectively does a *proc: b T*.

Miscellaneous commands:

**!command**

The command is interpreted by *sh(1)*.

**new-line**

If the previous command printed a source line, then advance the current line by one line and print the new current line. If the previous command displayed a memory location, then display the next memory location.

**control-D**

Scroll. Print the next 10 lines of instructions, source or data depending on which was printed last.

**< filename**

Read commands from *filename* until the end of file is reached, and then continue to accept commands from standard input. When *sdb* is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; **<** may not appear as a command in a file.

**M** Print the address maps.

**M [?/|\*] b e f**

Record new values for the address map. For the iAPX286, the parameters will be interpreted as follows:

- ?** represents **a.out** image
- /** represents **core** image
- \*** will be ignored
- b** will be an address interpreted as in the address section dependent on the file type; the bottom 16 bits will normally be zero.
- e** will be the extent size and can cover more than 64k bytes, as this will modify the required number of the array elements.
- f** will be the file store address for the start of the area.

**" string**

Print the given string. The C escape sequences of the form *\character* are recognized, where *character* is a nonnumeric character.

**q** Exit the debugger.

The following commands also exist and are intended only for debugging the debugger:

**V** Print the version number.

**Q** Print a list of procedures and files being debugged.

**Y** Toggle debug output.

**FILES**

a.out  
core

**SEE ALSO**

cc(1), f77(1), sh(1).

a.out(4), core(4) in the Software Development System manual.

## SDB(1)

### WARNINGS

User variables which differ by only an initial underscore cannot be distinguished, as *sdb* recognizes both internal and external names.

When *sdb* prints the value of an external variable for which there is no debugging information, a warning is printed before the value. The value is assumed to be *int* (integer).

Data which are stored in text sections are indistinguishable from functions.

Line number information in optimized functions is unreliable, and some information may be missing.

### BUGS

If a procedure is called when the program is *not* stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

The default type for printing F77 parameters is incorrect. Their address is printed instead of their value.

Tracebacks containing F77 subprograms with multiple entry points may print too many arguments in the wrong order, but their values are correct.

The range of an F77 array subscript is assumed to be *l* to *n*, where *n* is the dimension corresponding to that subscript. This is only significant when the user omits a subscript, or uses \* to indicate the full range. There is no problem in general with arrays having subscripts whose lower bounds are not 1.

On the 3B20 computer there is no hardware trace mode and single stepping is implemented by setting pseudo breakpoints where possible. This is slow. The *s*, *S*, *i*, and *I* commands do not always convert on the 3B20 computer due to pseudo-breakpointing. Thus *sdb* will not allow single-stepping from an *indirect* jump, a *switch* instruction, or a *switdt* instruction.

The entry point to an optimized function cannot be found on the 3B20 computer. Setting a breakpoint at the beginning of an optimized function may cause the middle of some instruction within the function to be overwritten. This problem can be circumvented by disassembling the first few instructions of the function, and manually setting a breakpoint at the first instruction after the stack pointer is adjusted.

**NAME**

sdiff -- side-by-side difference program

**SYNOPSIS**

sdiff [ options ... ] file1 file2

**DESCRIPTION**

*Sdiff* uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```

x | y
a | a
b <
c <
d | d
 > c

```

The following options exist:

- w *n*     Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.
- l         Only print the left side of any lines that are identical.
- s         Do not print identical lines.
- o *output* Use the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed commands:

```

l append the left column to the output file
r append the right column to the output file
s turn on silent mode; do not print identical lines
v turn off silent mode
e l call the editor with the left column
e r call the editor with the right column
e b call the editor with the concatenation of left and
 right
e call the editor with a zero length file
q exit from the program

```

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

**SEE ALSO**

diff(1), ed(1).

## SED(1)

### NAME

sed - stream editor

### SYNOPSIS

sed [ -n ] [ -e script ] [ -f sfile ] [ files ]

### DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **-f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **-e** option and no **-f** options, the flag **-e** may be omitted. The **-n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[ address [ , address ] ] function [ arguments ]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **-n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **\$** that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of *ed(1)* modified thus:

In a context address, the construction *?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address *\xabc\xdfx*, the second *x* stands for itself, so that the regular expression is *abcxdf*.

The escape sequence *\n* matches a new-line *embedded* in the pattern space.

A period *.* matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with *\* to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

- (1) a) *text* Append. Place *text* on the output before reading the next input line.
- (2) b *label* Branch to the : command bearing the *label*. If *label* is empty, branch to the end of the script.
- (2) c) *text* Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.
- (2) d Delete the pattern space. Start the next cycle.
- (2) D Delete the initial segment of the pattern space through the first new-line. Start the next cycle.
- (2) g Replace the contents of the pattern space by the contents of the hold space.
- (2) G Append the contents of the hold space to the pattern space.
- (2) h Replace the contents of the hold space by the contents of the pattern space.
- (2) H Append the contents of the pattern space to the hold space.
- (1) i) *text* Insert. Place *text* on the standard output.
- (2) I List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.
- (2) n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.
- (2) N Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)
- (2) p Print. Copy the pattern space to the standard output.
- (2) P Copy the initial segment of the pattern space through the first new-line to the standard output.
- (1) q Quit. Branch to the end of the script. Do not start a new cycle.
- (2) r *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.
- (2) s/*regular expression/replacement/flags*  
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of */*. For a fuller description see *ed(1)*. *Flags* is zero or more of:
- |                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| n              | n = 1 - 512. Substitute for just the n th occurrence of the <i>regular expression</i> .                              |
| g              | Global. Substitute for all nonoverlapping instances of the <i>regular expression</i> rather than just the first one. |
| p              | Print the pattern space if a replacement was made.                                                                   |
| w <i>wfile</i> | Write. Append the pattern space to <i>wfile</i> if a replacement was made.                                           |
- (2) t *label* Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a t. If *label* is empty, branch to the end of the script.
- (2) w *wfile* Write. Append the pattern space to *wfile*.
- (2) x Exchange the contents of the pattern and hold spaces.
- (2) y/*string1/string2/*  
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

## SED(1)

### (2)! *function*

Don't. Apply the *function* (or group, if *function* is `()`) only to lines *not* selected by the address(es).

(0): *label* This command does nothing; it bears a *label* for **b** and **t** commands to branch to.

(1) = Place the current line number on the standard output as a line.

(2) { Execute the following commands through a matching `}` only when the pattern space is selected.

(0) An empty command is ignored.

(0) # If a `#` appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the `#` is an `'n'`, then the default output will be suppressed. The rest of the line after `#n` is also ignored. A script file must contain at least one non-comment line.

## SEE ALSO

awk(1), ed(1), grep(1).

## NAME

send, gath — gather files and/or submit RJE jobs

## SYNOPSIS

gath [-ih] file ...

send argument ...

## DESCRIPTION

## Gath

*Gath* concatenates the named files and writes them to the standard output. Tabs are expanded into spaces according to the format specification for each file (see *fspec(4)*). The size limit and margin parameters of a format specification are also respected. Non-graphic characters other than tabs are identified by a diagnostic message and excised. The output of *gath* contains no tabs unless the *-b* flag is set, in which case the output is written with standard tabs (every eighth column).

Any line of any of the files which begins with *~* is interpreted by *gath* as a control line. A line beginning "*~* " (tilde,space) specifies a sequence of files to be included at that point. A line beginning *!* specifies a UNIX system command; that command is executed, and its output replaces the *!* line in the *gath* output.

Setting the *-i* flag prevents control lines from being interpreted and causes them to be output literally.

A file name of *-* at any point refers to standard input, and a control line consisting of *~* is a logical EOF. Keywords may be defined by specifying a replacement string which is to be substituted for each occurrence of the keyword. Input may be collected directly from the terminal, with several alternatives for prompting. In fact, all of the special arguments and flags recognized by the *send* command are also recognized and treated identically by *gath*. Several of them only make sense in the context of submitting an RJE job.

## Send

*Send* is a command-level interface to the RJE subsystems. It allows the user to collect input from various sources in order to create a run stream consisting of card images, and submit this run stream for transmission to an IBM host computer. Output from the IBM system may be returned to the user in either ASCII text form or EBCDIC punch format (see *pnch(4)*). How output is to be disposed of once it returns from the host is determined by a "usr=" specification which should be embedded in each job that a user submits for transmission. A detailed description of RJE operation and the "usr=" specification is given in *UNIX System Remote Job Entry User Guide*.

Possible sources of input to *send* are: ordinary files, standard input, the terminal, and the output of a command or shell file. Each source of input is treated as a virtual file, and no distinction is made based upon its origin. Typical input is an ASCII text file of the sort that is created by the editor *ed(1)*. An optional format specification appearing in the first line of a file (see *fspec(4)*) determines the settings according to which tabs are expanded into spaces. In addition, lines that begin with *~* are normally interpreted as commands controlling the execution of *send*. They may be used to set or reset flags, to define keyword substitutions, and to open new sources of input in the midst of the current source. Other text lines are translated one-for-one into card images of the run stream.

The run stream that results from this collection is treated as one job by the RJE subsystems. *Send* prints the card count of the run stream, and the queuer that is invoked prints the name of the temporary file that holds the job while it is awaiting transmission. The initial card of a job submitted to a host must have

## SEND(1C)

a // in the first column. Any cards preceding this card will be excised. If a host computer is not specified before the first card of the runstream is ready to be sent, *send* will select a reasonable default. All cards beginning with /\*\$ will be excised from the runstream, because they are HASP command cards.

The arguments that *send* accepts are described below. An argument is interpreted according to the first pattern that it matches. Preceding a character with \ causes it to lose any special meaning it might otherwise have when matching against an argument pattern.

|                 |                                                                                                                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .               | Close the current source.                                                                                                                                                                           |
| --              | Open standard input as a new source.                                                                                                                                                                |
| +               | Open the terminal as a new source.                                                                                                                                                                  |
| :spec:          | Establish a default format specification for included sources,<br>e.g., :m6t-12:                                                                                                                    |
| :message        | Print message on the terminal.                                                                                                                                                                      |
| :-prompt        | Open standard input and, if it is a terminal, print <i>prompt</i> .                                                                                                                                 |
| +:prompt        | Open the terminal and print <i>prompt</i> .                                                                                                                                                         |
| --flags         | Set the specified flags, which are described below.                                                                                                                                                 |
| +flags          | Reset the specified flags.                                                                                                                                                                          |
| =flags          | Restore the specified flags to their state at the previous level.                                                                                                                                   |
| !command        | Execute the specified UNIX system <i>command</i> via the one-line shell, with input redirected to /dev/null as a default. Open the standard output of the command as a new source.                  |
| \$line          | Collect contiguous arguments of this form and write them as consecutive lines to a temporary file; then have the file executed by the shell. Open the standard output of the shell as a new source. |
| @directory      | The current directory for the send process is changed to <i>directory</i> . The original directory will be restored at the end of the current source.                                               |
| ~comment        | Ignore this argument.                                                                                                                                                                               |
| ?keyword        | Prompt for a definition of <i>keyword</i> from the terminal unless <i>keyword</i> has an existing definition.                                                                                       |
| ?keyword=^xx    | Define the <i>keyword</i> as a two-digit hexadecimal character code unless it already has a non-null replacement.                                                                                   |
| ?keyword=string | Define the <i>keyword</i> in terms of a replacement string unless it already has a non-null replacement.                                                                                            |
| =:keyword       | Prompt for a definition of <i>keyword</i> from the terminal.                                                                                                                                        |
| keyword=^xx     | Define <i>keyword</i> as a two-digit hexadecimal character code.                                                                                                                                    |
| keyword=string  | Define <i>keyword</i> in terms of a replacement string.                                                                                                                                             |



*host* The host machine that the job should be submitted to. It can be any name that corresponds to one in the first column of the RJE configuration file (*/usr/rje/lines*).

*file-name* Open the specified file as a new source of input.

When commands are executed via **\$** or **!** the shell environment [see *environ(5)*] will contain the values of all send keywords that begin with **\$** and have the syntax of a shell variable.

The flags recognized by *send* are described in terms of the special processing that occurs when they are set:

- l List card images on standard output. EBCDIC characters are translated back to ASCII.
- q Do not output card images.
- f Do not fold lowercase to upper.
- t Trace progress on diagnostic output, by announcing the opening of input sources.
- k Ignore the keywords that are active at the previous level and erase any keyword definitions that have been made at the current level.
- r Process included sources in raw mode; pack arbitrary 8-bit bytes one per column (80 columns per card) until an EOF.
- i Do not interpret control lines in included sources; treat them as text.
- s Make keyword substitutions before detecting and interpreting control lines.
- y Suppress error diagnostics and submit job anyway.
- g Gather mode, qualifying -l flag; list text lines before converting them to card images.
- h Write listing with standard tabs.
- p Prompt with \* when taking input from the terminal.
- m When input returns to the terminal from a lower level, repeat the prompt, if any.
- a Make -k flag propagate to included sources, thereby protecting them from keyword substitutions.
- c List control lines on diagnostic output.
- d Extend the current set of keyword definitions by adding those active at the end of included sources.
- x This flag guarantees that the job will be transmitted in the order of submission (relative to other jobs sent with this flag).

Control lines are input lines that begin with **~**. In the default mode **+ir**, they are interpreted as commands to *send*. Normally they are detected immediately and read literally. The **-s** flag forces keyword substitutions to be made before control lines are intercepted and interpreted. This can lead to unexpected results if a control line uses a keyword which is defined within an immediately preceding **~\$** sequence. Arguments appearing in control lines are handled exactly like the command arguments to *send*, except that they are processed at a nested level of input.

The two possible formats for a control line are: "**~argument**" and "**~##argument#...**". In the first case, where the **~** is not followed by a space, the remainder of the line is taken as a single argument to *send*. In

the second case, the line is parsed to obtain a sequence of arguments delimited by spaces. In this case the quotes ' and " may be employed to pass embedded spaces.

The interpretation of the argument . is chosen so that an input line consisting of . is treated as a logical EOF. The following example illustrates some of the above conventions:

```
send## -
~##argument ...
.
```

This sequence of three lines is equivalent to the command synopsis at the beginning of this description. In fact, the - is not even required. By convention, the *send* command reads standard input if no other input source is specified. *Send* may therefore be employed as a filter with side-effects.

The execution of the *send* command is controlled at each instant by a current environment, which includes the format specification for the input source, a default format specification for included sources, the settings of the mode flags, and the active set of keyword definitions. This environment can be altered dynamically. When a control line opens a new source of input, the current environment is pushed onto a stack, to be restored when input resumes from the old source. The initial format specification for the new source is taken from the first line of the file. If none is provided, the established default is used or, in its absence, standard tabs. The initial mode settings and active keywords are copied from the old environment. Changes made while processing the new source will not affect the environment of the old source, with one exception: if -d mode is set in the old environment, the old keyword context will be augmented by those definitions that are active at the end of the new source.

When *send* first begins execution, all mode flags are reset, and the values of the shell environment variables become the initial values for keywords of the same name with a \$ prefixed.

The initial reset state for all mode flags is the + state. In general, special processing associated with a mode *N* is invoked by flag -*N* and is revoked by flag +*N*. Most mode settings have an immediate effect on the processing of the current source. Exceptions to this are the -r and -i flags, which apply only to included source, causing it to be processed in an uninterpreted manner.

A keyword is an arbitrary 8-bit ASCII string for which a replacement has been defined. The replacement may be another string or the hexadecimal code for a single 8-bit byte. At any instant, a given set of keyword definitions is active. Input text lines are scanned, in one pass from left to right, and longest matches are attempted between substrings of the line and the active set of keywords. Characters that do not match are output, subject to folding and the standard translation. Keywords are replaced by the specified hexadecimal code or replacement string, which is then output character by character. The expansion of tabs and length checking, according to the format specification of an input source, are delayed until substitutions have been made in a line.

All of the keywords definitions made in the current source may be deleted by setting the -k flag. It then becomes possible to reuse them. Setting the -k flag also causes keyword definitions active at the previous source level to be ignored. Setting the +k flag causes keywords at the previous level to be ignored but does not delete the definitions made at the current level. The =k argument reactivates the definitions of the previous level.

When keywords are redefined, the previous definition at the same level of source input is lost, however the definition at the previous level is only hidden, to be reactivated upon return to that level unless a `-d` flag causes the current definition to be retained.

Conditional prompts for keywords, `?A,/p` which have already been defined at some higher level to be null or have a replacement will simply cause the definitions to be copied down to the current level; new definitions will not be solicited.

Keyword substitution is an elementary macro facility that is easily explained and that appears useful enough to warrant its inclusion in the `send` command. More complex replacements are the function of a general macro processor `m4(1)`, perhaps. To reduce the overhead of string comparison, it is recommended that keywords be chosen so that their initial characters are unusual. For example, let them all be uppercase.

`Send` performs two types of error checking on input text lines. Primarily, only ASCII graphics and tabs are permitted in input text. Secondly, the length of a text line, after substitutions have been made, may not exceed 80 bytes. The length of each line may be additionally constrained by a size parameter in the format specification for an input source. Diagnostic output provides the location of each erroneous line, by line number and input source, a description of the error, and the card image that results. Other routine errors that are announced are the inability to open or write files, and abnormal exits from the shell. Normally, the occurrence of any error causes `send`, before invoking the queuer, to prompt for positive affirmation that the suspect run stream should be submitted.

Before submitting a job to a host, `send` translates 8-bit ASCII characters into their EBCDIC equivalents. The conversion for 8-bit ASCII characters in the octal range 040-176 is based on the character set described in "Appendix H" of *IBM System/370 Principles of Operation* (IBM SRL GA22-7000). Each 8-bit ASCII character in the range 040-377 possesses an EBCDIC equivalent into which it is mapped, with five exceptions: `#` into `~`, 0345 into `~`, 0325 into `¢`, 0313 into `}`, 0177 (DEL) is illegal. In listings requested from `send` and in printed output returned by the subsystem, the reverse translation is made with the qualification that EBCDIC characters that do not have valid 8-bit ASCII equivalents are translated into `~`.

Additional control over the translation process is afforded by the `-f` flag and hexadecimal character codes. As a default, `send` folds lowercase letters into uppercase. Setting the `-f` flag inhibits any folding. Non-standard character codes are obtained as a special case of keyword substitution. The users should check with the remote IBM system to be sure the special processing will be accepted.

#### SEE ALSO

`m4(1)`, `rjstat(1C)`, `sh(1)`.  
`lseek(2)`, `lspec(4)`, `pnch(4)`, `ascii(5)`, `environ(5)` in the Software Development System manual.

#### BUGS

Standard input is read in blocks, and unused bytes are returned via `lseek(2)`. If standard input is a pipe, multiple arguments of the form `-` and `~:prompt` should not be used, nor should the logical EOF (`~`).

# SETCOLOR(1)

## NAME

*setcolor* - Set foreground and background colors

## SYNOPSIS

*setcolor* [ [ -bBfF ] color1 ] [ -bBfF ] color2

## DESCRIPTION

*Setcolor* sets the text mode foreground and background colors on the Color, Enhanced, and Professional Graphics Adapters.

*Setcolor* -b denotes a dark background color, *setcolor* -B a bright one, -f a dark foreground color, and -F a bright one.

If no options, color1 is assumed to be a dark background color, and color2 is assumed to be a dark foreground color. Thus, *setcolor* color is assumed to mean *setcolor* -f color, and *setcolor* color1 color2 is assumed to mean *setcolor* -b color -f color.

The available colors are: black, red, green, yellow, brown, blue, maganta, cyan and white.

**NAME**  
setmnt — establish mount table

**SYNOPSIS**  
/etc/setmnt

**DESCRIPTION**

*Setmnt* creates the */etc/mnttab* table [see *mnttab(4)*], which is needed for both the *mount(1M)* and *umount* commands. *Setmnt* reads standard input and creates a *mnttab* entry for each line. Input lines have the format:

filesys node

where *filesys* is the name of the file system's *special file* (e.g., "dsk/?s?") and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the *mnttab(4)* entry.

**FILES**  
/etc/mnttab

**SEE ALSO**  
mount(1M), mnttab(4).

**BUGS**  
Evil things will happen if *filesys* or *node* are longer than 32 characters. *Setmnt* silently enforces an upper limit on the maximum number of *mnttab* entries.

## SETUP(1M)

### NAME

setup — change CMOS parameters

### SYNOPSIS

setup [-d] [keyword [options] ]

### DESCRIPTION

*Setup* lists or changes the permanent parameters in the PC/AT's CMOS ram, and also changes the time-of-day clock.

Here is a list of all available keywords and their effects:

|                       |                                                                |
|-----------------------|----------------------------------------------------------------|
| setup -d              | print date & time in date(1) format                            |
| setup all             | show all current values                                        |
| setup base            | prints Kbytes of base memory                                   |
| setup base 512        | sets base memory to 512 Kbytes<br>(assumes OK extended memory) |
| setup base 640        | sets base memroy to 640 Kbytes                                 |
| setup ext             | prints Kbytes of extended memory                               |
| setup ext number      | sets extended memory to number Kbytes                          |
| setup date            | prints current date setting                                    |
| setup date mm/dd/yy   | sets date to mm/dd/yy                                          |
| setup date mm/dd/yyyy | sets date to mm/dd/yyyy                                        |
| setup time            | prints current time setting                                    |
| setup time hh:mm:ss   | sets time to hh:mm:ss                                          |
| setup flop            | prints current floppy configuration                            |
| setup flop 1 low/high | sets first floppy drive to 48/96 tpi                           |
| setup flop 2 none     | sets no second floppy drive                                    |
| setup flop 2 low/high | sets second floppy drive to 48/96 tpi                          |
| setup fixed           | prints current fixed disk configuration                        |
| setup fixed 1 [1-99]  | sets first fixed drive to type 1-99                            |
| setup fixed 2 none    | sets second fixed drive as not installed                       |
| setup fixed 2 [1-99]  | sets second fixed drive to type 1-99                           |
| setup con             | prints current console                                         |
| setup con mono        | sets console to monochrome adapter, 80x25                      |
| setup con colr40      | sets console to color adapter, 40x25                           |
| setup con colr80      | sets console to color adapter, 80x25                           |
| setup con ega         | sets console to enhanced adapter, 80x25                        |

*Setup* prints out the old setting, the new setting, and then "Are you sure? (y orn)." The user must type y or Y. *Setup* then either does the change, prints "Setup confirmed" and exits, or prints "Setup aborted" and exits.

The command string

```
date 'setup -d'
```

sets the UNIX software clock to the date and time in the battery-operated real-time clock. This is not done automatically, as the real-time clocks in many AT-compatible machines are quite unreliable.

**NAME**

sh, rsh - shell, the standard/restricted command programming language

**SYNOPSIS**

```
sh [-acefhiknrstuvx] [args]
rsh [-acefhiknrstuvx] [args]
```

**DESCRIPTION**

*Sh* is a command programming language that executes commands read from a terminal or a file. *Rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

**Definitions**

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters \*, @, #, ?, -, \$, and !.

**Commands**

A *simple-command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 [see *exec(2)*]. The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally [see *signal(2)* for a list of status values].

A *pipeline* is a sequence of one or more *commands* separated by | (or, for historical compatibility, by ^). The standard output of each command but the last is connected by a *pipe(2)* to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by ;, &, &&, or ||, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol && (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

**for** *name* [ *in word ...* ] **do** *list* **done**

Each time a **for** command is executed, *name* is set to the next *word* taken from the *in word* list. If *in word ...* is omitted, then the **for** command executes the *do list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

**case** *word* **in** [ *pattern* [ | *pattern* ] ... ) *list* ;; ] ... **esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see *File Name Generation*) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

**if list then list { elif list then list } ... [ else list ] fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else list** is executed. If no **else list** or **then list** is executed, then the **if** command returns a zero exit status.

**while list do list done**

A **while** command repeatedly executes the *while list* and, if the exit status of the last command in the list is zero, executes the *do list*; otherwise the loop terminates. If no commands in the *do list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

**(list)**

Execute *list* in a sub-shell.

**{list;}**

*list* is simply executed.

**name () {list;}**

Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

**if then else elif fi case esac for while until do done { }**

#### Comments

A word beginning with **#** causes that word and all the following characters up to a new-line to be ignored.

#### Command Substitution

The standard output from a command enclosed in a pair of grave accents (``) may be used as part or all of a word; trailing new-lines are removed.

#### Parameter Substitution

The character **\$** is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

**name=value [ name=value ] ...**

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

**\$(parameter)**

The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is **\*** or **@**, all the positional parameters, starting with **\$1**, are substituted (separated by spaces). Parameter **\$0** is set from argument zero when the shell is invoked.

**\$(parameter:—word)**

If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

**\$(parameter:=word)**

If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.



**\$(parameter?:word)**

If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

**\$(parameter:+word)**

If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

- #** The number of positional parameters in decimal.
- Flags supplied to the shell on invocation or by the **set** command.
- ?** The decimal value returned by the last synchronously executed command.
- \$** The process number of this shell.
- !** The process number of the last background command invoked.

The following parameters are used by the shell:

**HOME** The default argument (home directory) for the **cd** command.  
**PATH** The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rsh*.

**CDPATH**

The search path for the **cd** command.

**MAIL** If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

**MAILCHECK**

This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

**MAILPATH**

A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is *you have mail*.

**PS1** Primary prompt string, by default "\$ "

**PS2** Secondary prompt string, by default "> "

**IFS** Internal field separators, normally **space**, **tab**, and **new-line**.

**SHACCT**

If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. Accounting routines such as *acctcom*(1) and *acctcms*(1M) can be used to analyze the data collected.

## SH(1)

**SHELL** When the shell is invoked, it scans the environment (see *Environment* below) for this name. If it is found and there is an 'r' in the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS**. **HOME** and **MAIL** are set by *login*(1).

### Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments (" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

### File Name Generation

Following substitution, each command *word* is scanned for the characters \*, ?, and |. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

- Matches any string, including the null string.
- ? Matches any single character.
- [...] Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "!" any character not enclosed is matched.

### Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > new-line space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair \new-line is ignored. All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, ', ", and \$. "\$\*" is equivalent to "\$1 \$2 ...", whereas "\$@" is equivalent to "\$1" "\$2" ....

### Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

### Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simple-command* or may precede or follow a *command* and are not passed on to the invoked command; substitution occurs before *word* or *digit* is used:

- <word Use file *word* as standard input (file descriptor 0).
- >word Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.
- >>word Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

- <<[ - ]word** The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) *\new-line* is ignored, and *\* must be used to quote the characters *\*, *\$*, *`*, and the first character of *word*. If *-* is appended to *<<*, all leading tabs are stripped from *word* and from the document.
- <& digit** Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using *>&digit*.
- <&-** The standard input is closed. Similarly for the standard output using *>&-*.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

```
... 1>xxx 2>&1
```

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e. *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

If a command is followed by *&* the default standard input for the command is the empty file */dev/null*. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

## Environment

The *environment* (see *environ(5)*) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the *export* command is used to bind the shell's parameter to the environment (see also *set -a*). A parameter may be removed from the environment with the *unset* command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by *unset*, plus any modifications or additions, all of which must be noted in *export* commands.

The environment for any *basb -r* command is executed (see below).

## Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

- :** No effect; the command does nothing. A zero exit code is returned.
- . file** Read and execute commands from *file* and return. The search path specified by *PATH* is used to find the directory containing *file*.

- break** [ *n* ]  
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.
- continue** [ *n* ]  
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.
- cd** [ *arg* ]  
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*. The **cd** command may not be executed by *rsh*.
- echo** [ *arg ...* ]  
Echo arguments. See **echo(1)** for usage and description.
- eval** [ *arg ...* ]  
The arguments are read as input to the shell and the resulting command(s) executed.
- exec** [ *arg ...* ]  
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.
- exit** [ *n* ]  
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)
- export** [ *name ...* ]  
The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. Function names may *not* be exported.
- hash** [ **-r** ] [ *name ...* ]  
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (\*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.
- newgrp** [ *arg ...* ]  
Equivalent to **exec newgrp arg ...**. See **newgrp(1)** for usage and description.  
The given *names* are marked *readonly* and the values of the these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.
- return** [ *n* ]  
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

**set** [ **--aefhknrtuvx** [ *arg* ... ] ]

- a** Mark variables which are modified or created for export.
- e** Exit immediately if a command exits with a non-zero exit status.
- f** Disable file name generation
- h** Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
- k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- n** Read commands but do not execute them.
- t** Exit after reading and executing one command.
- u** Treat unset variables as an error when substituting.
- v** Print shell input lines as they are read.
- x** Print commands and their arguments as they are executed.
- Do not change any of the flags; useful in setting **\$1** to **-**.

Using **+** rather than **-** causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **\$-**. The remaining arguments are positional parameters and are assigned, in order, to **\$1**, **\$2**, .... If no arguments are given the values of all names are printed.

**shift** [ *n* ]

The positional parameters from **\$n+1** ... are renamed **\$1** .... If *n* is not given, it is assumed to be 1.

**test**

Evaluate conditional expressions. See *test*(1) for usage and description.

**times**

Print the accumulated user and system times for processes run from the shell.

**trap** [ *arg* ] [ *n* ] ...

The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

**type** [ *name* ... ]

For each *name*, indicate how it would be interpreted if used as a command name.

**ulimit** [ **-fp** ] [ *n* ]

imposes a size limit of *n*

**-f** imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.

**-p** changes the pipe size to *n* (UNIX System/RT only).

If no option is given, **-f** is assumed.

**umask** [ *nnn* ]

The user file-creation mask is set to *nnn* [see *umask*(2)]. If *nnn* is omitted, the current value of the mask is printed.

**unset** [ *name ...* ]

For each *name*, remove the corresponding variable or function. The variables **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS** cannot be unset.

**wait** [ *n* ]

Wait for the specified process and report its termination status. If *n* is not given all currently active child processes are waited for and the return code is zero.

### Invocation

If the shell is invoked through *exec(2)* and the first character of argument zero is **-**, commands are initially read from **/etc/profile** and from **\$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/bin/sh**. The flags below are interpreted by the shell on invocation only; Note that unless the **-c** or **-s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

- c** *string* If the **-c** flag is present commands are read from *string*.
- s** If the **-s** flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.
- i** If the **-i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case **TERM** is ignored (so that **kill 0** does not kill an interactive shell) and **INTERRUPT** is caught and ignored (so that **wait** is interruptible). In all cases, **QUIT** is ignored by the shell.
- r** If the **-r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

### Rsh Only

*Rsh* is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

- changing directory [see *cd(1)*],
- setting the value of **\$PATH**,
- specifying path or command names containing **/**,
- redirecting output (**>** and **>>**).

The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

**EXIT STATUS**

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the `exit` command above).

**FILES**

`/etc/profile`  
`$HOME/.profile`  
`/tmp/sh*`  
`/dev/null`

**SEE ALSO**

`acctcom(1)`, `cd(1)`, `echo(1)`, `env(1)`, `login(1)`, `newgrp(1)`, `pwd(1)`, `test(1)`, `umask(1)`.

`dup(2)`, `exec(2)`, `fork(2)`, `pipe(2)`, `signal(2)`, `ulimit(2)`, `umask(2)`, `wait(2)`, `a.out(4)`, `profile(4)`, `environ(5)` in the Software Development System manual.

**CAVEATS**

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the `hash` command to correct this situation.

If you move the current directory or one above it, `pwd` may not give the correct response. Use the `cd` command with a full path name to correct this situation.

# SHL(1)

## NAME

shl - shell layer manager

## SYNOPSIS

shl

## DESCRIPTION

*Shl* allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as *layers*, using the commands described below.

The *current layer* is the layer which can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To have the output of a layer blocked when it is not current, the *stty* option *loblk* may be set within the layer.

The *stty* character *switch* (set to `^Z` if NUL) is used to switch control to *shl* from a layer. *Shl* has its own prompt, `>>>>`, to help distinguish it from a layer.

A *layer* is a shell which has been bound to a virtual tty device (`/dev/sxt???`). The virtual device can be manipulated like a real tty device using *stty*(1) and *ioctl*(2). Each layer has its own process group id.

## Definitions

A *name* is a sequence of characters delimited by a blank, tab or new-line. Only the first eight characters are significant. The *names* (1) through (7) cannot be used when creating a layer. They are used by *shl* when no name is supplied. They may be abbreviated to just the digit.

## Commands

The following commands may be issued from the *shl* prompt level. Any unique prefix is accepted.

**create** [ *name* ]

Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form (#) where # is the last digit of the virtual device bound to the layer. The shell prompt variable `PS1` is set to the name of the layer followed by a space. A maximum of seven layers can be created.

**block** *name* [ *name* ... ]

For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option *loblk* within the layer.

**delete** *name* [ *name* ... ]

For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the `SIGHUP` signal [see *signal*(2)].

**help** (or ?)

Print the syntax of the *shl* commands.

**layers** { `-l` } [ *name* ... ]

For each *name*, list the layer name and its process group. The `-l` option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

**resume** [ *name* ]

Make the layer referenced by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

**toggle** Resume the layer that was current before the last current layer.



**unblock** *name* [ *name ...* ]

For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **loblk** within the layer.

**quit** Exit *shl*. All layers are sent the SIGHUP signal.

*name* Make the layer referenced by *name* the current layer.

#### FILES

/dev/sxt??? Virtual tty devices

\$SHELL Variable containing path name of the shell to use (default is /bin/sh).

#### SEE ALSO

sh(1), stty(1).

ioctl(2), signal(2) in the Software Development System manual.

sxt(7) in the Runtime System manual.

Microport System V/AT does not currently support SHL. However, we plan to support it in the future.

# SHMCREATE(1)

## NAME

shmcreate – user-mode access to console graphics

## SYNOPSIS

shmcreate key address size

## DESCRIPTION

*Shmcreate* allows direct access to the screen memory section of PC/AT graphics adapter. It builds a shared memory segment (see *shmctl(2)*, *shmget(2)*, and *shmunp(2)* in the Software Development Manual Part II) which is mapped to a given physical address. It is only intended for bit-mapped screen use, but could theoretically be used as part of the memory management for an intelligent peripheral. For example, to make the default page of Hercules graphics available, run:

```
shmcreate 0xb0000 b0000 32768
```

This creates a segment whose "key" is hex B0000, mapped to physical address hex B0000, and whose length is 32768 bytes. This corresponds to the first screen on a Hercules graphics adapter. To make all the IBM cards available, place this in file */etc/rc.d/shm.rc*:

```
/etc/shmcreate 0xa0000 a0000 65535 # ega high res
/etc/shmcreate 0xb0000 b0000 65535 # ega low res
/etc/shmcreate 0xb8000 b8000 32768 # cga
/etc/shmcreate 0xc8000 c8000 4096 # pga
```

## EXAMPLE

Shmcreate must be run by the super-user. After it is run, any program may contain the following three lines to access, for example, the CGA:

```
extern char *shmat();
int shmids = shmget (0xb8000, 32768, 0);
char *shmidr = shmat (shmids, (char *) 0);
```

The *shmget* "opens" the physically mapped segment created by *shmcreate*. The *shmget* line returns an address which allows the user program to access the graphics memory on the screen adapter. On the Hercules board, the pointer refers to a line of dots at the upper left-hand corner of the screen. Refer to the Hercules manual for addressing information.

## NOTES

The key value does not need to be the same as the physical address. This is simply a convenience to avoid conflicts with other shared memory keys. These keys are also referenced by Microport's graphics programs.

This program cannot be run unless shared memory is supported in your system. The shared-memory scheme for screen access is being upgraded and the application developer should assume that it will be replaced, in the future, by a more graphics-specific approach.

**BUGS**

A segment which is 64 K long cannot be created, thus the last byte of the screen memory may not be used.

**SEE ALSO**

`shmctl(2)`, `shmget(2)`, `shmop(2)` in the Software Development Manual, Part II.

The user should also consult the programming manual available for the particular card being used. IBM's manuals are in the Options and Adapters Manual set.

## SHOWBAD(1M)

### NAME

showbad — display bad track table for hard disk partition

### SYNOPSIS

**showbad** [-1]

### DESCRIPTION

*Showbad* with no options displays bad track table information for the System V/AT partition on the primary drive. *Showbad* 1 displays the table for the secondary hard disk, unit 1.

The bad track table is created by option 5 of the fixed disk utility [see *fdisk(1M)*]. The bad track table lists cylinder number and head (track) for each track with one or more errors together with alternate cylinder number and head to be used in place of the bad track, when that track is encountered.

## NAME

shutdown - shut down system

## SYNOPSIS

*/etc/shutdown* [ -y ] [ -g*grace-period* [ -i*init-state* ]

## DESCRIPTION

This command is executed by the super-user, "root", to change the state of the machine. By default, it brings the system to a state where only the console has access to the UNIX system. This "root-only" state is traditionally called "single-user".

The command sends a warning message and a final message before it starts actual shutdown activities. By default, the command asks for confirmation before it starts shutting down daemons and killing processes. The options are used as follows:

- y pre-answers the question so the command can be run without user intervention. By default, the time periods between the warning message and the final message and between the final message and the confirmation are 60 seconds.
- g*grace-period* allows the super-user to specify a different number of seconds.
- i*init-state* specifies the state that *init*(1M) is to be put in following the warnings, if any. By default, init state "s" is used.

NOTE: THIS VERSION OF SHUTDOWN IS DIFFERENT FROM PREVIOUS VERSIONS.

In the past, the *shutdown* procedure performed process-killing and file system unmounts before changing init state. This proved unreliable. Now, the new init state defines what state the machine is to be in and is responsible for making it that way. Recommended definitions are:

- state 0  
Shut the machine down so it is safe to remove the power. Have the machine remove power if it can.
- state 2  
Bring machine to state traditionally called multiuser.
- state 5  
Stop the UNIX system and go to the firmware monitor.
- state 6  
Stop the UNIX system and reboot.

## SEE ALSO

*init*(1M).

## SIZE(1)

### NAME

size - print section sizes of common object files

### SYNOPSIS

size [-o] [-x] [-d] [-V] files

### DESCRIPTION

The *size* command produces section size information for each section in the object files in the common object file format. If an archive file is input to *size*, the information for all archive members is displayed.

The size of the *.text*, *.data*, and *.bss* (uninitialized data) sections are printed along with the total size of the object file. If there are sections other than *.text*, *.data*, and *.bss* that are loaded when the program is executed, the name of each section will be printed along with its size.

Numbers will be printed in decimal unless either the *-o* or the *-x* option is used, in which case they will be printed in octal or in hexadecimal, respectively.

The *-V* flag will supply the version information on the *size* command.

### SEE ALSO

as(1), cc(1), ld(1),  
a.out(4), ar(4) in the Software Development System manual.

### DIAGNOSTICS

"size: name: cannot open"  
if *name* cannot be read.

"size: name: bad magic"  
if *name* is not a common object file.

**NAME**  
sleep — suspend execution for an interval

**SYNOPSIS**  
sleep time

**DESCRIPTION**  
*Sleep* suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
 command
 sleep 37
done
```

**SEE ALSO**  
alarm(2), sleep(3C) in the Software Development Manual

## SNO(1)

### NAME

sno - SNOBOL interpreter

### SYNOPSIS

sno [ files ]

### DESCRIPTION

*Sno* is a SNOBOL compiler and interpreter (with slight differences). *Sno* obtains input from the concatenation of the named *files* and the standard input. All input through a statement containing the label **end** is considered program and is compiled. The rest is available to **syspit**.

*Sno* differs from SNOBOL in the following ways:

There are no unanchored searches. To get the same effect:

```
a ** b unanchored search for b.
a *x* b = x c unanchored assignment
```

There is no back referencing.

```
x = "abc"
a *x* x is an unanchored search for abc.
```

Function declaration is done at compile time by the use of the (non-unique) label **define**. Execution of a function call begins at the statement following the **define**. Functions cannot be defined at run time, and the use of the name **define** is preempted. There is no provision for automatic variables other than parameters. Examples:

```
define f()
define f(a, b, c)
```

All labels except **define** (even **end**) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on **end** cannot merely name a label.

If **start** is a label in the program, program execution will start there. If not, execution begins with the first executable statement; **define** is not an executable statement.

There are no built-in functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators **/** and **\*** must be set off by spaces.

The right side of assignments must be non-empty.

Either **'** or **"** may be used for literal quotes.

The pseudo-variable **syspit** is not available.

### SEE ALSO

awk(1).



**NAME**

sort - sort and/or merge files

**SYNOPSIS**

sort [-**cmu**] [-**o**output] [-**ykmem**] [-**zrecsz**] [-**dMnr**] [-**btx**] [+pos] [-pos2] [files]

**DESCRIPTION**

*Sort* sorts lines of all the named files together and writes the result on the standard output. The standard input is read if - is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The following options alter the default behavior:

- c** Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.
- m** Merge only, the input files are already sorted.
- u** Unique: suppress all but one in each set of lines having equal keys.

**-o**output

The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between -o and output.

**-ykmem**

The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, -y0 is guaranteed to start with minimum memory. By convention, -y (with no argument) starts with maximum memory.

**-zrecsz**

The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the -c or -m options, a popular system default size will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules.

- d** "Dictionary" order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.
- f** Fold lower case letters into upper case.
- i** Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.
- M** Compare as months. The first three non-blank characters of the field are folded to upper case and compared so that "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The -M option implies the -b option (see below).
- n** An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by

## SORT(1)

arithmetic value. The **-n** option implies the **-b** option (see below). Note that the **-b** option is only effective when restricted sort key specifications are in effect.

**-r** Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation **+pos1 -pos2** restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing **-pos2** means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

**-tx** Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field).

**-b** Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **-b** option is specified before the first **+pos1** argument, it will be applied to all **+pos1** arguments. Otherwise, the **b** flag may be attached independently to each **+pos1** or **-pos2** argument (see below).

*pos1* and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfnr**. A starting position specified by **+m.n** is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means *.0*, indicating the first character of the *m*+1st field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m*+1st field; **+m.0b** refers to the first non-blank character in the *m*+1st field.

A last position specified by **-m.n** is interpreted to mean the *n*th character (including separators) after the last character of the *m*th field. A missing *.n* means *.0*, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*+1st field; **-m.1b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

### EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 -2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort -r -o outfile +1.0 -1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

```
sort -r +1.0b -1.1b infile1 infile2
```

Print the password file [*passwd*(4)] sorted by the numeric user ID (the third colon-separated field):

```
sort -t: +2n -3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **-um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort -um +2 -3 infile
```

#### FILES

/usr/tmp/stm???

#### SEE ALSO

comm(1), join(1), uniq(1).

#### DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions (e.g., when input lines are too long), and for disorder discovered under the **-c** option. When the last line of an input file is missing a **new-line** character, *sort* appends one, prints a warning message, and continues.

## SPELL(1)

### NAME

spell, hashmake, spellin, hashcheck -- find spelling errors

### SYNOPSIS

```
spell [-v] [-b] [-x] [-l] [-i] [+local_file] [files]
/usr/lib/spell/hashmake
/usr/lib/spell/spellin n
/usr/lib/spell/hashcheck spelling_list
```

### DESCRIPTION

*Spell* collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

*Spell* ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

Under the *-v* option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the *-b* option, British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

Under the *-x* option, every plausible stem is printed with = for each word.

By default, *spell* [like *deroff*(1)] follows chains of included files [*.so* and *.nx troff*(1) requests], unless the names of such included files begin with */usr/lib*. Under the *-l* option, *spell* will follow the chains of all included files. Under the *-i* option, *spell* will ignore all chains of included files.

Under the *+local\_file* option, words found in *local\_file* are removed from *spell*'s output. *Local\_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., *thier=thy-y+tier*) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell*:

- |                  |                                                                                                                                                                                 |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>hashmake</b>  | Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.                                                         |
| <b>spellin n</b> | Reads <i>n</i> hash codes from the standard input and writes a compressed spelling list on the standard output. Information about the hash coding is printed on standard error. |
| <b>hashcheck</b> | Reads a compressed <i>spelling_list</i> and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.                          |

**EXAMPLES**

The following example creates the hashed spell list `hlist` and checks the result by comparing the two temporary files; they should be equal.

```
cat goodwds | /usr/lib/spell/hashmake | sort -u >tmp1
cat tmp1 | /usr/lib/spell/spellin `cat tmp1 | wc -l` >hlist
cat hlist | /usr/lib/spell/hashcheck >tmp2
diff tmp1 tmp2
```

**FILES**

|                                               |                                           |
|-----------------------------------------------|-------------------------------------------|
| <code>D_SPELL=/usr/lib/spell/hlist[ab]</code> | hashed spelling lists, American & British |
| <code>S_SPELL=/usr/lib/spell/hstop</code>     | hashed stop list                          |
| <code>H_SPELL=/usr/lib/spell/spellhist</code> | history file                              |
| <code>/usr/lib/spell/spellprog</code>         | program                                   |

**SEE ALSO**

`sed(1)`, `sort(1)`, `tee(1)`.

"Nroff and Troff User manual", "Mathematics Typesetting Program" (`egn`), and "Table formatting Program" (`tbl`) in the Text Preparation System manual.

**BUGS**

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling\_list* via *spellin*.

The British spelling feature was done by an American.

## SPLIT(1)

### NAME

split — split a file into pieces

### SYNOPSIS

split [ -n ] [ file [ name ] ]

### DESCRIPTION

*Split* reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with *aa* appended, and so on lexicographically, up to *zz* (a maximum of 676 files). *Name* cannot be longer than 12 characters. If no output name is given, *x* is default.

If no input file is given, or if *-* is given in its stead, then the standard input file is used.

### SEE ALSO

bfs(1), csplit(1).

## NAME

strip - strip symbol and line number information from common object files

## SYNOPSIS

strip [-b] [-l] [-r] [-s] [-x] [-V] file-names

## DESCRIPTION

The *strip* command strips the symbol table and line number information from common object files, including archives. Once this has been done, no symbolic debugging access will be available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using the following options:

- b Same as the -x option but also do not strip scoping information (i.e., beginning and end of block delimiters).
- l Strip line number information only; do not strip any symbol table information.
- r Reset the relocation indices into the symbol table.
- s Reset the line number indices into the symbol table (do not remove). Reset the relocation indices into the symbol table.
- x Do not strip static or external symbol information.
- V Print the version of the *strip* command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, *strip* will complain and terminate without stripping *file-name* unless the -r flag is used.

If the *strip* command is executed on an archive file [see *ar(4)*] the archive symbol table will be removed. The archive symbol table must be restored by executing the *ar(1)* command with the s option before the archive can be link-edited by the *ld(1)* command. *Strip(1)* will instruct the user with appropriate warning messages when this situation arises.

The purpose of this command is to reduce the file storage overhead taken by the object file.

## FILES

/usr/tmp/str?????

## SEE ALSO

*ar(1)*, *as(1)*, *cc(1)*, *ld(1)*.  
*a.out(4)* in the Software Development System manual.

## DIAGNOSTICS

- strip: name: cannot open  
           if *name* cannot be read.
- strip: name: bad magic  
           if *name* is not a common object file.
- strip: name: relocation entries present; cannot strip  
           if *name* contains relocation entries and the -r flag is not used,  
           and any symbol table information was to be stripped.

**This page intentionally left blank.**



## NAME

stty — set the options for a terminal

## SYNOPSIS

stty [ -a ] [ -g ] [ options ]

## DESCRIPTION

*Stty* sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options; with the **-a** option, it reports all of the option settings; with the **-g** option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first five groups below may be found in *termio(7)* for asynchronous lines, or in *stermio(7)* for synchronous lines. Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

## Control Modes

**parneb (-parneb)** enable (disable) parity generation and detection.  
**parodd (-parodd)** select odd (even) parity.  
**cs5 cs6 cs7 cs8** select character size [see *termio(7)*].  
**0** hang up phone line immediately.  
**50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb** Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)  
**bupcl (-bupcl)** hang up (do not hang up) DATA-PHONE® connection on last close.  
**bup (-bup)** same as **bupcl (-bupcl)**.  
**cstopb (-cstopb)** use two (one) stop bits per character.  
**cread (-cread)** enable (disable) the receiver.  
**local (-local)** n assume a line without (with) modem control.  
**loblk (-loblk)** block (do not block) output from a non-current layer.

## Input Modes

**ignbrk (-ignbrk)** ignore (do not ignore) break on input.  
**brkint (-brkint)** signal (do not signal) INTR on break.  
**ignpar (-ignpar)** ignore (do not ignore) parity errors.  
**parmrk (-parmrk)** mark (do not mark) parity errors [see *termio(7)*].  
**inpck (-inpck)** enable (disable) input parity checking.  
**istrip (-istrip)** strip (do not strip) input characters to seven bits.  
**inlcr (-inlcr)** map (do not map) NL to CR on input.  
**igncr (-igncr)** ignore (do not ignore) CR on input.  
**icrnl (-icrnl)** map (do not map) CR to NL on input.  
**iucL (-iucL)** map (do not map) uppercase alphabetic to lower case on input.  
**ixon (-ixon)** enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.  
**ixany (-ixany)** allow any character (only DC1) to restart output.  
**ixoff (-ixoff)** request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

## Output Modes

**opost (-opost)** post-process output (do not post-process output; ignore all other output modes).  
**olcuc (-olcuc)** map (do not map) lowercase alphabetic to uppercase on output.

## STTY(1)

|                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>onlcr</b> ( <b>-onlcr</b> )                   | map (do not map) NL to CR-NL on output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>ocrnl</b> ( <b>-ocrnl</b> )                   | map (do not map) CR to NL on output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>onocr</b> ( <b>-onocr</b> )                   | do not (do) output CRs at column zero.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>onlret</b> ( <b>-onlret</b> )                 | on the terminal NL performs (does not perform) the CR function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>ofill</b> ( <b>-ofill</b> )                   | use fill characters (use timing) for delays.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>ofdel</b> ( <b>-ofdel</b> )                   | fill characters are DELs (NULs).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>cr0 cr1 cr2 cr3</b>                           | select style of delay for carriage returns [see <i>termio(7)</i> ].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>nl0 nl1</b>                                   | select style of delay for line-feeds [see <i>termio(7)</i> ].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>tab0 tab1 tab2 tab3</b>                       | select style of delay for horizontal tabs [see <i>termio(7)</i> or <i>stermio(7)</i> ].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>bs0 bs1</b>                                   | select style of delay for backspaces [see <i>termio(7)</i> ].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>ff0 ff1</b>                                   | select style of delay for form-feeds [see <i>termio(7)</i> ].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>vt0 vt1</b>                                   | select style of delay for vertical tabs [see <i>termio(7)</i> ].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Local Modes</b>                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>isig</b> ( <b>-isig</b> )                     | enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>icanon</b> ( <b>-icanon</b> )                 | enable (disable) canonical input (ERASE and KILL processing).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>xcase</b> ( <b>-xcase</b> )                   | canonical (unprocessed) upper/lowercase presentation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>echo</b> ( <b>-echo</b> )                     | echo back (do not echo back) every character typed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>echoe</b> ( <b>-echoe</b> )                   | echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEd character on many CRT terminals; however, it does <i>not</i> keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.                                                                                                                                                                                                                                                                                                |
| <b>echok</b> ( <b>-echok</b> )                   | echo (do not echo) NL after KILL character.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>lfkc</b> ( <b>-lfkc</b> )                     | the same as <b>echok</b> ( <b>-echok</b> ); obsolete.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>echonl</b> ( <b>-echonl</b> )                 | echo (do not echo) NL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>noflsh</b> ( <b>-noflsh</b> )                 | disable (enable) flush after INTR, QUIT, or SWTCH.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>stwrap</b> ( <b>-stwrap</b> )                 | disable (enable) truncation of lines longer than 79 characters on a synchronous line.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>stflush</b> ( <b>-stflush</b> )               | enable (disable) flush on a synchronous line after every <i>write(2)</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>stappl</b> ( <b>-stappl</b> )                 | use application mode (use line mode) on a synchronous line.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Control Assignments</b>                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>control-character c</i>                       | set <i>control-character</i> to <i>c</i> , where <i>control-character</i> is <b>erase</b> , <b>kill</b> , <b>intr</b> , <b>quit</b> , <b>swtch</b> , <b>eof</b> , <b>ctab</b> , <b>min</b> , or <b>time</b> [ <b>ctab</b> is used with <b>-stappl</b> ; see <i>stermio(7)</i> ], [ <b>min</b> and <b>time</b> are used with <b>-icanon</b> ; see <i>termio(7)</i> ]. If <i>c</i> is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., “^d” is a CTRL-d); “^?” is interpreted as DEL and “^_” is interpreted as undefined. |
| <b>line i</b>                                    | set line discipline to <i>i</i> ( $0 < i < 127$ ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Combination Modes</b>                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>evenp</b> or <b>parity</b>                    | enable <b>parenb</b> and <b>cs7</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>oddp</b>                                      | enable <b>parenb</b> , <b>cs7</b> , and <b>parodd</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>-parity</b> , <b>-evenp</b> , or <b>-oddp</b> | disable <b>parenb</b> , and set <b>cs8</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>raw</b> ( <b>-raw</b> or <b>cooked</b> )      | enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

|                                             |                                                                                                                                                                                  |
|---------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>nl</b> ( <b>-nl</b> )                    | unset (set) <b>icrnl</b> , <b>onlcr</b> . In addition <b>-nl</b> unsets <b>inlcr</b> , <b>igncr</b> , <b>ocrnl</b> , and <b>onlret</b> .                                         |
| <b>lcase</b> ( <b>-lcase</b> )              | set (unset) <b>xcase</b> , <b>iucLc</b> , and <b>olcuc</b> .                                                                                                                     |
| <b>LCASE</b> ( <b>-LCASE</b> )              | same as <b>lcase</b> ( <b>-lcase</b> ).                                                                                                                                          |
| <b>tabs</b> ( <b>-tabs</b> or <b>tab3</b> ) | preserve (expand to spaces) tabs when printing.                                                                                                                                  |
| <b>ek</b>                                   | reset ERASE and KILL characters back to normal # and @.                                                                                                                          |
| <b>sane</b>                                 | resets all modes to some reasonable values.                                                                                                                                      |
| <b>term</b>                                 | set all modes suitable for the terminal type <i>term</i> , where <i>term</i> is one of <b>tty33</b> , <b>tty37</b> , <b>vt05</b> , <b>tn300</b> , <b>ti700</b> , or <b>tek</b> . |

**SEE ALSO**

**tabs**(1).  
**ioctl**(2) in the Software Development System manual.  
**sternio**(7), **termio**(7).

## SU(1)

### NAME

**su** — become super-user or another user

### SYNOPSIS

**su** [ - ] [ name [ arg ... ] ]

### DESCRIPTION

*Su* allows one to become another user without logging off. The default user name is **root** (i.e., super-user).

To use *su*, the appropriate password must be supplied (unless one is already **root**). If the password is correct, *su* will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry [see *passwd*(4)], or **/bin/sh** if none is specified [see *sh*(1)]. To restore normal user ID privileges, type an EOF (*ctrl-d*) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like *sh*(1), an *arg* of the form **-c string** executes *string* via the shell and an *arg* of **-r** will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(1). If the first argument to *su* is a **-**, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is **-**, thus causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of **\$PATH**, which is set to **/bin:/etc:/usr/bin** for **root**. Note that if the optional program used as the shell is **/bin/sb**, the user's **.profile** can check *arg0* for **-sb** or **-su** to determine if it was invoked by *login*(1) or *su*(1), respectively. If the user's program is other than **/bin/sb**, then **.profile** is invoked with an *arg0* of **-program** by both *login*(1) and *su*(1).

All attempts to become another user using *su* are logged in the log file **/usr/adm/sulog**.

### EXAMPLES

To become user **bin** while retaining your previously exported environment, execute:

```
su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, execute:

```
su - bin
```

To execute *command* with the temporary environment and permissions of user **bin**, type:

```
su - bin -c "command args"
```

## FILES

|                 |                        |
|-----------------|------------------------|
| /etc/passwd     | system's password file |
| /etc/profile    | system's profile       |
| \$HOME/.profile | user's profile         |
| /usr/adm/sulog  | log file               |

## SEE ALSO

env(1), login(1), sh(1).  
passwd(4), profile(4), environ(5) in the Software Development System manual.

## SUM(1)

### NAME

sum - print checksum and block count of a file

### SYNOPSIS

sum [ -r ] file

### DESCRIPTION

*Sum* calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **-r** causes an alternate algorithm to be used in computing the checksum.

### SEE ALSO

wc(1).

### DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

**NAME**

sync — update the super block

**SYNOPSIS**

sync

**DESCRIPTION**

*Sync* executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync(2)* for details.

**SEE ALSO**

*sync(2)* in the Software Development System manual.

## **SYSDEF(1M)**

### **NAME**

*sysdef* — system definition

### **SYNOPSIS**

*/etc/sysdef* [ *opsys* [ *master* ] ]

### **DESCRIPTION**

*Sysdef* analyzes the named operating system file and extracts configuration information. This includes all hardware devices as well as system devices and all tunable parameters.

The output of *sysdef* can usually be used directly by *config(1M)* to regenerate the appropriate configuration files.

### **FILES**

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>/unix</i>       | default operating system file             |
| <i>/etc/master</i> | default table for hardware specifications |

### **SEE ALSO**

*config(1M)*, *master(4)*.

### **BUGS**

For devices that have interrupt vectors but are not interrupt-driven, the output of *sysdef* cannot be used for *config*. Because information regarding *config* aliases is not preserved by the system, device names returned might not be accurate.



## NAME

tabs — set tabs on a terminal

## SYNOPSIS

tabs [ tabspec ] [ +mn ] [ -Ttype ]

## DESCRIPTION

*Tabs* sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

Users of GE TermiNet terminals should be aware that they behave in a different way than most other terminals for some tab settings. The first number in a list of tab settings becomes the *left margin* on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

Four types of tab specification are accepted for *tabspec*: "canned," repetitive, arbitrary, and file. If no *tabspec* is given, the default value is -8, i.e., UNIX system "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

-code Gives the name of one of a set of "canned" tabs. The legal codes and their meanings are as follows:

- a 1,10,16,36,72  
Assembler, IBM S/370, first format
- a2 1,10,16,40,72  
Assembler, IBM S/370, second format
- c 1,8,12,16,20,55  
COBOL, normal format
- c2 1,6,10,14,49  
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:  
<:t-c2 m6 s66 d:>
- c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67  
COBOL compact format (columns 1-6 omitted), with more tabs than -c2. This is the recommended format for COBOL. The appropriate format specification is:  
<:t-c3 m6 s66 d:>
- f 1,7,11,15,19,23  
FORTRAN
- p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61  
PL/I
- s 1,10,55  
SNOBOL
- u 1,12,20,44  
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

- n A repetitive specification requests tabs at columns  $1+n$ ,  $1+2*n$ , etc. Note that such a setting leaves a left margin of  $n$  columns on TermiNet terminals *only*. Of particular importance is the value -8: this represents the UNIX system "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with

## TABS(1)

the *nroff* **-h** option for high-speed output. Another special case is the value **-0**, implying no tabs at all.

*n1,n2,...* The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

**--file** If the name of a file is given, *tabs* reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(1) command:  
tabs **--** file; *pr* file

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

**-Ttype** *Tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed in *term*(5). If no **-T** flag is supplied, *tabs* searches for the **STERM** value in the *environ* [see *environ*(5)]. If no *type* can be found, *tabs* tries a sequence that will work for many terminals.

**+mn** The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

### DIAGNOSTICS

|                          |                                                                                                                                        |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>illegal tabs</i>      | when arbitrary tabs are ordered incorrectly.                                                                                           |
| <i>illegal increment</i> | when a zero or missing increment is found in an arbitrary specification.                                                               |
| <i>unknown tab code</i>  | when a "canned" code cannot be found.                                                                                                  |
| <i>can't open</i>        | if <b>--file</b> option used, and file can't be opened.                                                                                |
| <i>file indirection</i>  | if <b>--file</b> option used and the specification in that file points to yet another file. Indirection of this form is not permitted. |

### SEE ALSO

*pr*(1).  
*environ*(5), *term*(5) in the Software Development System manual.

### BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

It is generally impossible to usefully change the left margin without also setting tabs.

*Tabs* clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

**NAME**

`tail` — deliver the last part of a file

**SYNOPSIS**

`tail [ ±[number][lbcf] ] [ file ]`

**DESCRIPTION**

*Tail* copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *-number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option *l*, *b*, or *c*. When no units are specified, counting is by lines.

With the *-f* ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail -f fred
```

will print the last ten lines of the file `fred`, followed by any lines that are appended to `fred` between the time *tail* is initiated and killed. As another example, the command:

```
tail -15cf fred
```

will print the last 15 characters of the file `fred`, followed by any lines that are appended to `fred` between the time *tail* is initiated and killed.

**SEE ALSO**

`dd(1)`.

**BUGS**

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

## TAR(1)

### NAME

tar - tape file archiver

### SYNOPSIS

tar [ key ] [ files ]

### DESCRIPTION

*Tar* saves and restores files on magnetic tape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the *key* is specified by one of the following letters:

- r** The named *files* are written on the end of the tape. The **c** function implies this function.
- x** The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. If a named file on tape does not exist on the system, the file is created with the same mode as the one on tape except that the set-user-ID and set-group-ID bits are not set unless you are super-user. If the files exist, their modes are not changed except for the bits described above. The owner, group, and modification time are restored (if possible). If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.
- t** The names of all the files on the tape are listed.
- u** The named *files* are added to the tape if they are not already there, or have been modified since last written on that tape.
- c** Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This command implies the **r** function.

The following characters may be used in addition to the letter that selects the desired function:

- #s** Where **#** is a tape drive number (0,...,7), and **s** is the density (l - low (800 bpi), m - medium (1600 bpi), or h - high (6250 bpi)). This modifier selects the drive on which the tape is mounted. The default is 0m.
- v** Normally, *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.
- w** Causes *tar* to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with **y** is given, the action is performed. Any other input means "no".
- f** Causes *tar* to use the next argument as the name of the archive instead of /dev/mt/???. If the name of the file is -, *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *Tar* can also be used to move hierarchies with the command:

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

- b** Causes *tar* to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes (key letters **x** and **t**).
- l** Tells *tar* to complain if it cannot resolve all of the links to the files being dumped. If **l** is not specified, no error messages are printed.
- m** Tells *tar* not to restore the modification times. The modification time of the file will be the time of extraction.
- o** Causes extracted files to take on the user and group identifier of the user running the program rather than those on the tape.

#### EXAMPLES

```
tar wvf /dev/rdisk/fd table of contents
 (high density floppy)

tar xvf /dev/rdisk/fd restore files to hard disk
```

#### FILES

```
/dev/mt/*
/tmp/tar*
```

#### DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.  
Complaints if enough memory is not available to hold the link tables.

#### BUGS

There is no way to ask for the *n*-th occurrence of a file.  
Tape errors are handled ungracefully.  
The **u** option can be slow.  
The **b** option should not be used with archives that are going to be updated.  
The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the **b** option should not be used at all, because updating an archive stored on disk can destroy it.  
The current limit on file-name length is 100 characters.  
Note that **tar c0m** is not the same as **tar cm0**.

## TEE(1)

### NAME

tee - pipe fitting

### SYNOPSIS

tee [ -i ] [ -a ] [ file ] ...

### DESCRIPTION

*Tee* transcribes the standard input to the standard output and makes copies in the *files*. The *-i* option ignores interrupts; the *-a* option causes the output to be appended to the *files* rather than overwriting them.

## NAME

test — condition evaluation command

## SYNOPSIS

test expr  
[ expr ]

## DESCRIPTION

*Test* evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

- r *file* true if *file* exists and is readable.
- w *file* true if *file* exists and is writable.
- x *file* true if *file* exists and is executable.
- f *file* true if *file* exists and is a regular file.
- d *file* true if *file* exists and is a directory.
- c *file* true if *file* exists and is a character special file.
- b *file* true if *file* exists and is a block special file.
- p *file* true if *file* exists and is a named pipe (fifo).
- u *file* true if *file* exists and its set-user-ID bit is set.
- g *file* true if *file* exists and its set-group-ID bit is set.
- k *file* true if *file* exists and its sticky bit is set.
- s *file* true if *file* exists and has a size greater than zero.
- t [ *filides* ] true if the open file whose file descriptor number is *filides* (1 by default) is associated with a terminal device.
- z *s1* true if the length of string *s1* is zero.
- n *s1* true if the length of the string *s1* is non-zero.
- s1* = *s2* true if strings *s1* and *s2* are identical.
- s1* != *s2* true if strings *s1* and *s2* are *not* identical.
- s1* true if *s1* is *not* the null string.
- n1* -eq *n2* true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons -ne, -gt, -ge, -lt, and -le may be used in place of -eq.

These primaries may be combined with the following operators:

- ! unary negation operator.
- a binary *and* operator.
- o binary *or* operator (-a has higher precedence than -o).
- ( *expr* ) parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

## TEST(1)

### SEE ALSO

find(1), sh(1).

### WARNING

In the second form of the command (i.e., the one that uses `[]`, rather than the word *test*), the square brackets must be delimited by blanks. Some UNIX systems do not recognize the second form of the command.



**NAME**

tic - terminfo compiler

**SYNOPSIS**

tic [ -v[n] ] file ...

**DESCRIPTION**

*Tic* translates terminfo files from the source format into the compiled format. The results are placed in the directory **/usr/lib/terminfo**.

The **-v** (verbose) option causes *tic* to output trace information showing its progress. If the optional integer is appended, the level of verbosity can be increased.

*Tic* compiles all terminfo descriptions in the given files. When a **use=** field is discovered, *tic* searches first the current file, then the master file, which is **"/terminfo.src"**.

If the environment variable **TERMINFO** is set, the results are placed there instead of **/usr/lib/terminfo**.

Some limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

**FILES**

**/usr/lib/terminfo/\*/\*** compiled terminal capability data base

**SEE ALSO**

**curses(3X)**, **terminfo(4)**.

**BUGS**

Instead of searching **/terminfo.src**, it should check for an existing compiled entry.

## TIME(1)

### NAME

time -- time a command

### SYNOPSIS

time command

### DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on standard error.

### SEE ALSO

timex(1).

times(2) in the Software Development System manual.

### CAVEATS

When *time* is used on a 3B20A dual computer system the sum of system and user time could be greater than real time. This is the result when *command* is a multi-threaded task running on a 3B20A system with both processors active.

**NAME**  
 timex — time a command; report process data and system activity

**SYNOPSIS**  
 timex [options] command

**DESCRIPTION**  
 The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of *timex* is written on standard error.

*Options* are:

- p List process accounting records for *command* and all its children. Suboptions *f*, *b*, *k*, *m*, *r*, and *t* modify the data items reported, as defined in *acctcom*(1). The number of blocks read or written and the number of characters transferred are always reported.
- o Report the total number of blocks read or written and total characters transferred by *command* and all its children.
- s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

**SEE ALSO**  
*acctcom*(1), *sar*(1).

**CAVEATS**  
 When *timex* is used on a 3B20A dual computer system the sum of system and user time could be greater than real time. This is the result when *command* is a multi-threaded task running on a 3B20A system with both processors active.

**WARNING**  
 Process records associated with *command* are selected from the accounting file */usr/adm/pacct* by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

**EXAMPLES**  
 A simple example:

```
timex -ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
timex -opskmt sh
 session commands
EOT
```

## TOUCH(1)

### NAME

`touch` — update access and modification times of a file

### SYNOPSIS

`touch [ -amc ] [ mmddhhmm[yy] ] files`

### DESCRIPTION

*Touch* causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified [see *date(1)*] the current time is used. The `-a` and `-m` options cause *touch* to update only the access or modification times respectively (default is `-am`). The `-c` option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

### SEE ALSO

*date(1)*,  
*utime(2)* in the Software Development System manual.

**NAME**

*tput* - query terminfo data base

**SYNOPSIS**

*tput* [ *-T*type ] capname

**DESCRIPTION**

*Tput* uses the *terminfo*(4) data base to make terminal-dependent capabilities and information available to the shell. *Tput* outputs a string if the attribute (*capability name*) is of type string, or an integer if the attribute is of type integer. If the attribute is of type Boolean, *tput* simply sets the exit code (0 for TRUE, 1 for FALSE), and does no output.

*-T*type indicates the type of terminal. Normally this flag is unnecessary, as the default is taken from the environment variable **\$TERM**.

*Capname* indicates the attribute from the *terminfo* database. See *terminfo*(4).

**EXAMPLES**

*tput clear* Echo clear-screen sequence for the current terminal.  
*tput cols* Print the number of columns for the current terminal.  
*tput -T450 cols* Print the number of columns for the 450 terminal.  
*bold=\$(tput smso)* Set shell variable "bold" to stand-out mode sequence for current terminal. This might be followed by a prompt:  
**echo "\${bold}Please type in your name: \c"**  
*tput hc* Set exit code to indicate if current terminal is a hardcopy terminal.

**FILES**

*/etc/term/!/\** Terminal descriptor files  
*/usr/include/term.h* Definition files  
*/usr/include/curses.h*

**DIAGNOSTICS**

*Tput* prints error messages and returns the following error codes on error:  
 -1 Usage error.  
 -2 Bad terminal type.  
 -3 Bad capname.

In addition, if a capname is requested for a terminal that has no value for that capname (e.g., *tput -T450 lines*), -1 is printed.

**SEE ALSO**

*stty*(1).  
*terminfo*(4) in the Software Development System manual.

## TR(1)

### NAME

tr - translate characters

### SYNOPSIS

tr [ -cds ] [ string1 [ string2 ] ]

### DESCRIPTION

*Tr* copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options *-cds* may be used:

- c* Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.
- d* Deletes all input characters in *string1*.
- s* Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

- [*a-z*] Stands for the string of characters whose ASCII codes run from character *a* to character *z*, inclusive.
- [*a\*n*] Stands for *n* repetitions of *a*. If the first digit of *n* is 0, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabets. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

```
tr -cs "[A-Z][a-z]" "\012*" <file1 >file2
```

### SEE ALSO

ed(1), sh(1).  
ascii(5) in the Software Development System manual.

### BUGS

Will not handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

**NAME**  
true, false — provide truth values

**SYNOPSIS**  
**true**  
**false**

**DESCRIPTION**  
*True* does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh(1)* such as:

```
while true
do
 command
done
```

**SEE ALSO**  
*sh(1)*.

**DIAGNOSTICS**  
*True* has exit status zero, *false* nonzero.

## TSORT(1)

### NAME

tsort - topological sort

### SYNOPSIS

tsort [ file ]

### DESCRIPTION

*Tsort* produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

### SEE ALSO

lorder(1).

### DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

### BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.



**NAME**

tty - get the name of the terminal

**SYNOPSIS**

tty [ -l ] [ -s ]

**DESCRIPTION**

*Tty* prints the path name of the user's terminal. The **-l** option prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line. The **-s** option inhibits printing of the terminal path name, allowing one to test just the exit code.

**EXIT CODES**

2 if invalid options were specified,  
0 if standard input is a terminal,  
1 otherwise.

**DIAGNOSTICS**

"not on an active synchronous line" if the standard input is not a synchronous terminal and **-l** is specified.

"not a tty" if the standard input is not a terminal and **-s** is not specified.

## TTYPATCH(1)

### NAME

ttypatch - patch a kernel for tty parameters

### SYNOPSIS

ttypatch [ -k cd ] [ -t (tty#) ] [ -i (interrupt) ] [ -a (addr) ]  
[ -n (count) ] [ -v (vector) ] [ -m (modem ctl value) ]

### DESCRIPTION

ttypatch makes dev/tty\* node(s) as needed and patches kernel parameters. By default, the filename /system5 is patched.

- k patches the dev/kmem (running) version of the kernel. Note that the disk file is not patched.
- c clears (zeros) the kernel parameters.
- d prints the current values of the kernel parameters.
- t# starting tty number (i.e. 0 for /dev/tty0).
- i# interrupt line number (i.e. 4 for COM1, 3 for COM2).
- a# address of 16450/8250 UART base (i.e. 1016 for COM1, 760 for COM2).
- n# number of consecutive UARTS (8 i/o locations apart) on this interrupt line.
- v# port address to enable interrupts, latch individual interrupt lines (0 if not necessary to read/write).
- m# Modem Control Register value (values for OUT1 & OUT2 are board/port dependent, default= 0x0c).

Numbers and addresses should be in decimal for expr compatibility.

### FILES

/dev/tty?/, /dev/ttyM?/, /dev/ttyM??

### SEE ALSO

patch(1), mknod(1)

### EXAMPLES

Standard COM1 (tty0 at int 4 and address 0x3f8):

```
"ttypatch -t0 -a1016 -i4"
```

Standard COM2 (tty1 at int 3 and address 0x2f8):

```
"ttypatch -t1 -a760 -i3"
```

8 port Digiboard using COM2's interrupt. Devices tty0 through tty7 are created:

```
"ttypatch -t0 -n8 -a256 -i3"
```

4 port Digiboard. This command assumes the link kit has been used to allow interrupt 5 for serial service. Devices tty2 through tty5 are created:

```
"ttypatch -t2 -i5 -a256 -n4"
```

4 port AST board. This command assumes the link kit has been used to allow interrupt 7 for serial service. Devices tty6 through tty9 are created:

```
"ttypatch -t6 -i7 -a416 -n4 -v447 -m0"
```

**NAME**

`uadmin` - administrative control

**SYNOPSIS**

`uadmin` cmd fcn

**DESCRIPTION**

The *uadmin* command provides control for basic administrative functions. This command is tightly coupled to the system administration procedures and is not intended for general use. It may be invoked only by the super-user.

The arguments are converted to integers and passed to the *uadmin* system call.

**SEE ALSO**

`uadmin(2)` in the Software Development System manual.

# UMASK(1)

## NAME

`umask` - set file-creation mode mask

## SYNOPSIS

`umask` [ *ooo* ]

## DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod(2)* and *umask(2)*). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat(2)*). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode 777 become mode 755; files created with mode 666 become mode 644).

If *ooo* is omitted, the current value of the mask is printed.

*Umask* is recognized and executed by the shell.

## SEE ALSO

*chmod(1)*, *sh(1)*.

*chmod(2)*, *creat(2)*, *umask(2)* in the Software Development System manual.

**NAME**

uname — print name of current UNIX system

**SYNOPSIS**

uname [ -snrvma ]

**DESCRIPTION**

*Uname* prints the current system name of the UNIX system on the standard output file. It is mainly useful to determine which system one is using. The options cause selected information returned by *uname(2)* to be printed:

- s print the system name (default).
- n print the nodename (the nodename may be a name that the system is known by to a communications network).
- r print the operating system release.
- v print the operating system version.
- m print the machine hardware name.
- a print all the above information.

**SEE ALSO**

*uname(2)* in the Software Development System manual.

## UNGET(1)

### NAME

`unget` — undo a previous `get` of an SCCS file

### SYNOPSIS

`unget` [`-rSID`] [`-s`] [`-n`] files

### DESCRIPTION

`Unget` undoes the effect of a `get -e` done prior to creating the intended new delta. If a directory is named, `unget` behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of `-` is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

- `-rSID`      Uniquely identifies which delta is no longer intended. (This would have been specified by `get` as the "new delta"). The use of this keyletter is necessary only if two or more outstanding `gets` for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified `SID` is ambiguous, or if it is necessary and omitted on the command line.
- `-s`              Suppresses the printout, on the standard output, of the intended delta's `SID`.
- `-n`              Causes the retention of the gotten file which would normally be removed from the current directory.

### SEE ALSO

`delta(1)`, `get(1)`, `help(1)`, `sact(1)`.

### DIAGNOSTICS

Use `help(1)` for explanations.

**NAME**            **uniq** — report repeated lines in a file

**SYNOPSIS**  
**uniq** [ **-udc** [ **+n** ] [ **-n** ] ] [ **input** [ **output** ] ]

**DESCRIPTION**

*Uniq* reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **-u** flag is used, just the lines that are not repeated in the original file are output. The **-d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **-u** and **-d** mode outputs.

The **-c** option supersedes **-u** and **-d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

- n**    The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.
- +n**    The first *n* characters are ignored. Fields are skipped before characters.

**SEE ALSO**  
*comm*(1), *sort*(1).

## UNITS(1)

### NAME

units — conversion program

### SYNOPSIS

units

### DESCRIPTION

*Units* converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
 • 2.540000e+00
 / 3.937008e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
 • 1.020689e+00
 / 9.797299e-01
```

*Units* only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

|              |                                         |
|--------------|-----------------------------------------|
| <b>pi</b>    | ratio of circumference to diameter,     |
| <b>c</b>     | speed of light,                         |
| <b>e</b>     | charge on an electron,                  |
| <b>g</b>     | acceleration of gravity,                |
| <b>force</b> | same as g,                              |
| <b>mole</b>  | Avogadro's number,                      |
| <b>water</b> | pressure head per unit height of water, |
| <b>au</b>    | astronomical unit.                      |

**Pound** is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

### FILES

```
/usr/lib/unittab
```



**NAME**

untic -- Uncompile terminfo terminal description files

**SYNOPSIS**

untic terminal-name ...

**DESCRIPTION**

*Untic* converts the terminfo file which corresponds to the specified terminal name into a file that can be processed by tic.

If multiple terminal names are given, a terminal description is generated for each of the named terminals. The output is written on standard output.

It is wise to backup the original terminal description (from */usr/lib/terminfo*) before compiling the generated file.

**EXAMPLES**

To uncompile the terminal description for a "vt100", type:

```
untic vt100 >vt100
```

To compile this generated file, you probably need root permissions and type:

```
tic vt100
```

**NOTES**

If new capabilities are added to the terminfo database, the untic program will need to be modified.

Most of the capabilities are in the tables in the order that the capabilities are documented. There are a few exceptions in the string variables.

**FILES**

```
/src/untic.c
/usr/lib/terminfo/?/*
```

**AUTHOR**

Dave Regan

**This page intentionally left blank.**

**NAME**

uuclean — uucp spool directory clean-up

**SYNOPSIS**

**/usr/lib/uucp/uuclean** [ options ]

**DESCRIPTION**

*Uuclean* will scan the spool directory for files with the specified prefix and delete all those which are older than the specified number of hours.

The following options are available.

- ddirectory** Clean *directory* instead of the spool directory. If *directory* is not a valid spool directory it cannot contain "work files" i.e., files whose names start with "C.". These files have special meaning to *uuclean* pertaining to *uucp* job statistics.
- ppre** Scan for files with *pre* as the file prefix. Up to 10 **-p** arguments may be specified. A **-p** without any *pre* following will cause all files older than the specified time to be deleted.
- ntime** Files whose age is more than *time* hours will be deleted if the prefix test is satisfied (default time is 72 hours).
- wfile** The default action for *uuclean* is to remove files which are older than a specified time (see **-n** option). The **-w** option is used to find those files older than *time* hours, however, the files are not deleted. If the argument *file* is present the warning is placed in *file*, otherwise, the warnings will go to the standard output.
- ssys** Only files destined for system *sys* are examined. Up to 10 **-s** arguments may be specified.
- mfile** The **-m** option sends mail to the owner of the file when it is deleted. If a *file* is specified then an entry is placed in *file*.

This program is typically started by *cron*(1M).

**FILES**

|                        |                                                           |
|------------------------|-----------------------------------------------------------|
| <b>/usr/lib/uucp</b>   | directory with commands used by <i>uuclean</i> internally |
| <b>/usr/spool/uucp</b> | spool directory                                           |

**SEE ALSO**

*cron*(1M), *uucp*(1C), *uux*(1C).

# UUCP(1C)

## NAME

uucp, uulog, uuname — UNIX system-to-UNIX system copy

## SYNOPSIS

**uucp** [ options ] source-files destination-file

**uulog** [ options ]

**uuname** [ -l ] [ -v ]

## DESCRIPTION

### Uucp.

*Uucp* copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names which *uucp* knows about. The *system-name* may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file via the specified route, and only to a destination in PUBDIR (see below). Care should be taken to insure that intermediate nodes in the route are willing to forward information.

The shell metacharacters `?`, `*` and `[...]` appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

- (1) a full path name;
- (2) a path name preceded by `~user` where *user* is a login name on the specified system and is replaced by that user's login directory;
- (3) a path name preceded by `~/user` where *user* is a login name on the specified system and is replaced by that user's directory under PUBDIR;
- (4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

*Uucp* preserves execute permissions across the transmission and gives 0666 read and write permissions [see *chmod(2)*].

The following options are interpreted by *uucp*:

- d** Make all necessary directories for the file copy (default).
- f** Do not make intermediate directories for the file copy.
- c** Use the source file when copying out rather than copying the file to the spool directory (default).
- C** Copy the source file to the spool directory.
- m file** Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.
- muser** Notify *user* on the remote system that a file was sent.
- esys** Send the *uucp* command to system *sys* to be executed there. (Note: this will only be successful if the remote machine allows the *uucp* command to be executed by `/usr/lib/uucp/uuxqt`.)

- r Queue job but do not start the file transfer process. By default a file transfer process is started each time uucp is evoked.
- j Control writing of the *uucp* job number to standard output (see below).

*Uucp* associates a job number with each request. This job number can be used by *uustat* to obtain status or terminate the job.

The environment variable **JOBNO** and the **-j** option are used to control the listing of the *uucp* job number on standard output. If the environment variable **JOBNO** is undefined or set to **OFF**, the job number will not be listed (default). If *uucp* is then invoked with the **-j** option, the job number will be listed. If the environment variable **JOBNO** is set to **ON** and is exported, a job number will be written to standard output each time *uucp* is invoked. In this case, the **-j** option will suppress output of the job number.

### Uulog

*Uulog* queries a summary log of *uucp* and *uux*(1C) transactions in the file */usr/spool/uucp/LOGFILE*.

The options cause *uulog* to print logging information:

- ssys Print information about work involving system *sys*. If *sys* is not specified, then logging information for all systems will be printed.
- user Print information about work done for the specified, *user*. If *user* is not specified then logging information for all users will be printed.

### Uuname.

*Uuname* lists the *uucp* names of known systems. The **-l** option returns the local system name. The **-v** option will print additional information about each system. A description will be printed for each system that has a line of information in */usr/lib/uucp/ADMIN*. The format of **ADMIN** is: *sysname* tab *description* tab.

### FILES

*/usr/spool/uucp* spool directory  
*/usr/spool/uucppublic* public directory for receiving and sending (PUBDIR)  
*/usr/lib/uucp/\** other data and program files

### SEE ALSO

mail(1), *uux*(1C).  
 chnod(2) in the Software Development System manual.

### WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons, you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin */usr/spool/uucppublic* (equivalent to *~uucp* or just *~*).

### NOTES

In order to send files that begin with a dot (e.g., *.profile*) the files must be qualified with a dot. For example: *.profile*, *.prof\**, *.profil?* are correct; whereas *\*prof\**, *?profile* are incorrect.

*Uucp* will not generate a job number for a strictly local transaction.

## UUCP(1C)

### BUGS

All files received by *uucp* will be owned by *uucp*.

The `-m` option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters `? * [...]` will not activate the `-m` option.

The `-m` option will not work if all transactions are local or if *uucp* is executed remotely via the `-e` option.

The `-m` option will function only when the source and destination are not on the same machine.

Only the first six characters of a *system-name* are significant. Any excess characters are ignored.

**NAME**

uustat — uucp status inquiry and job control

**SYNOPSIS**

**uustat** [ options ]

**DESCRIPTION**

*Uustat* will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. The following *options* are recognized:

- j***jobn* Report the status of the *uucp* request *jobn*. If **all** is used for *jobn*, the status of all *uucp* requests is reported. An argument must be supplied otherwise the usage message will be printed and the request will fail.
- k***jobn* Kill the *uucp* request whose job number is *jobn*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.
- r***jobn* Rejuvenate *jobn*. That is, *jobn* is touched so that its modification time is set to the current time. This prevents *uuclean* from deleting the job until the jobs modification time reaches the limit imposed by *uuclean*.
- c***hour* Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the super-user.
- u***user* Report the status of all *uucp* requests issued by *user*.
- s***sys* Report the status of all *uucp* requests which communicate with remote system *sys*.
- o***hour* Report the status of all *uucp* requests which are older than *hour* hours.
- y***hour* Report the status of all *uucp* requests which are younger than *hour* hours.
- m***mch* Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local *uucp* are provided.
- M***mch* This is the same as the **-m** option except that two times are printed. The time that the last status was obtained and the time that the last successful transfer to that system occurred.
- O** Report the *uucp* status using the octal status codes listed below. If this option is not specified, the verbose description is printed with each *uucp* request.
- q** List the number of jobs and other control files queued for each machine and the time of the oldest and youngest file queued for each machine. If a lock file exists for that system, its date of creation is listed.

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user. Note that only one of the options **-j**, **-m**, **-k**, **-c**, **-r**, can be used with the rest of the other options.

For example, the command:

```
uustat -uhdc -smhtsa -y72
```

will print the status of all *uucp* requests that were issued by user *hdc* to communicate with system *mhtsa* within the last 72 hours. The meanings of the job request status are:

job-number user remote-system command-time status-time status

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

## UUSTAT(1C)

| OCTAL  | STATUS                                               |
|--------|------------------------------------------------------|
| 000001 | the copy failed, but the reason cannot be determined |
| 000002 | permission to access local file is denied            |
| 000004 | permission to access remote file is denied           |
| 000010 | bad <i>uucp</i> command is generated                 |
| 000020 | remote system cannot create temporary file           |
| 000040 | cannot copy to remote directory                      |
| 000100 | cannot copy to local directory                       |
| 000200 | local system cannot create temporary file            |
| 000400 | cannot execute <i>uucp</i>                           |
| 001000 | copy (partially) succeeded                           |
| 002000 | copy finished, job deleted                           |
| 004000 | job is queued                                        |
| 010000 | job killed (incomplete)                              |
| 020000 | job killed (complete)                                |

The meanings of the machine accessibility status are:

system-name time status

where *time* is the latest status time and *status* is a self-explanatory description of the machine status.

### FILES

|                             |                     |
|-----------------------------|---------------------|
| <i>/usr/spool/uucp</i>      | spool directory     |
| <i>/usr/lib/uucp/L_stat</i> | system status file  |
| <i>/usr/lib/uucp/R_stat</i> | request status file |

### SEE ALSO

*uucp*(1C).



## NAME

uusub — monitor uucp network

## SYNOPSIS

`/usr/lib/uucp/uusub [ options ]`

## DESCRIPTION

`Uusub(1M)` defines a *uucp* subnetwork and monitors the connection and traffic among the members of the subnetwork. The following options are available:

- `-a sys` Add *sys* to the subnetwork.
- `-d sys` Delete *sys* from the subnetwork.
- `-l` Report the statistics on connections.
- `-r` Report the statistics on traffic amount.
- `-f` Flush the connection statistics.
- `-uhr` Gather the traffic statistics over the past *hr* hours.
- `-c sys` Exercise the connection to the system *sys*. If *sys* is specified as **all**, then exercise the connection to all the systems in the subnetwork.

The meanings of the connections report are:

`sys #call #ok time #dev #login #nack #other`

where *sys* is the remote system name, *#call* is the number of times the local system tries to call *sys* since the last flush was done, and *#ok* is the number of successful connections, *time* is the latest successful connect time, *#dev* is the number of unsuccessful connections because of no available device (e.g., ACU), *#login* is the number of unsuccessful connections because of login failure, *#nack* is the number of unsuccessful connections because of no response (e.g., line busy, system down), and *#other* is the number of unsuccessful connections because of other reasons.

The meanings of the traffic statistics are:

`sfile sbyte rfile rbyte`

where *sfile* is the number of files sent and *sbyte* is the number of bytes sent over the period of time indicated in the latest *uusub* command with the `-uhr` option. Similarly, *rfile* and *rbyte* are the numbers of files and bytes received.

The command:

`uusub -c all -u 24`

is typically started by *cron(1M)* once a day.

## FILES

|                                     |                       |
|-------------------------------------|-----------------------|
| <code>/usr/spool/uucp/SYSLOG</code> | system log file       |
| <code>/usr/lib/uucp/L_sub</code>    | connection statistics |
| <code>/usr/lib/uucp/R_sub</code>    | traffic statistics    |

## SEE ALSO

`uucp(1C)`, `uustat(1C)`.

## UUTO(1C)

### NAME

uuto, uupick - public UNIX-to-UNIX system file copy

### SYNOPSIS

**uuto** [ options ] source-files destination  
**uupick** [ -s system ]

### DESCRIPTION

*Uuto* sends *source-files* to *destination*. *Uuto* uses the *uucp*(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:  
system!user

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*). *Logname* is the login name of someone on the specified system.

Two *options* are available:

- p Copy the source file into the spool directory before transmission.
- m Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. Specifically the files are sent to

PUBDIR/receive/user!mysystem/files.

The destined recipient is notified by *mail*(1) of the arrival of files.

*Uupick* accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from system: [file file-name] [dir dirname] ?

*Uupick* then reads a line from the standard input to determine the disposition of the file:

- <new-line> Go on to next entry.
- d Delete the entry.
- m [ dir ] Move the entry to named directory *dir* (current directory is default).
- a [ dir ] Same as *m* except moving all the files sent from *system*.
- p Print the content of the file.
- q Stop.
- EOT (control-d) Same as *q*.
- !command Escape to the shell to do *command*.
- \* Print a command summary.

*Uupick* invoked with the *-ssystem* option will only search the PUBDIR for files sent from *system*.

### FILES

PUBDIR/usr/spool/uucppublic public directory

### NOTES

In order to send files that begin with a dot (e.g., .profile) the files must be qualified with a dot. For example: .profile, .prof\*, .profil? are correct; whereas \*prof\*, ?profile are incorrect.

### SEE ALSO

mail(1), uucp(1C), uustat(1C), uux(1C),  
uuclean(1M).

## NAME

*uux* - UNIX-to-UNIX system command execution

## SYNOPSIS

*uux* [ options ] *command-string*

## DESCRIPTION

*Uux* will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail (see *mail(1)*) via *uux*.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of

- (1) a full path name;
- (2) a path name preceded by `~xxx` where *xxx* is a login name on the specified system and is replaced by that user's login directory;
- (3) anything else is prefixed by the current directory.

As an example, the command

```
uux "diff usg!usr/dan/fl pwba!a4/dan/fl > !fl.diff"
```

will get the *fl* files from the "usg" and "pwba" machines, execute a *diff* command and put the results in *fl.diff* in the local directory.

Any special shell characters such as `<>|` should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

*Uux* will attempt to get all files to the execution system. For files which are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!uucp b!usr/file \c!usr/file\
```

will send a *uucp* command to system "a" to get */usr/file* from system "b" and send it to system "c".

*Uux* will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine. Executable commands are listed in */usr/lib/uucp/L.cmds* on the remote system. The format of the *L.cmds* file is:

```
cmd,machine1,machine2,...
```

If no machines are specified, then any machine can execute *cmd*. If machines are specified, only the listed machines can execute *cmd*. If the desired command is not listed in *L.sys* then no machine can execute that command.

Redirection of standard input and output is usually restricted to files in PUB-DIR. Directories into which redirection is allowed must be specified in */usr/lib/uucp/USERFILE* by the system administrator. See "UUCP."

The following *options* are interpreted by *uux*:

- The standard input to *uux* is made the standard input to the *command-string*.
- n Send no notification to user.

## UUX(1C)

**-mfile** Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

**-j** Control writing of the *uucp* job number to standard output.

*Uux* associates a job number with each request. This job number can be used by *uustat* to obtain status or terminate the job.

The environment variable **JOBNO** and the **-j** option are used to control the listing of the *uux* job number on standard output. If the environment variable **JOBNO** is undefined or set to **OFF**, the job number will not be listed (default). If *uuco* is then invoked with the **-j** option, the job number will be listed. If the environment variable **JOBNO** is set to **ON** and is exported, a job number will be written to standard output each time *uux* is invoked. In this case, the **-j** option will suppress output of the job number.

### FILES

|                              |                           |
|------------------------------|---------------------------|
| <i>/usr/spool/uucp</i>       | spool directory           |
| <i>/usr/spool/uucppublic</i> | public directory (PUBDIR) |
| <i>/usr/lib/uucp/*</i>       | other data and programs   |

### SEE ALSO

*mail(1)*, *uuclean(1M)*, *uucp(1C)*.

### BUGS

Only the first command of a shell pipeline may have a *system-name!*. All other commands are executed on the system of the first command.

The use of the shell metacharacter *\** will probably not do what you want it to do. The shell tokens *<<* and *>>* are not implemented.

Only the first six characters of the *system-name* are significant. Any excess characters are ignored.

## NAME

val - validate SCCS file

## SYNOPSIS

val -  
val [-s] [-rSID] [-mname] [-ytype] files

## DESCRIPTION

*Val* determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a -, and named files.

*Val* has a special argument, -, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

*Val* generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

- |        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -s     | The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.                                                                                                                                                                                                                                                                           |
| -rSID  | The argument value <i>SID</i> (SCCS <i>I</i> Dentification String) is an SCCS delta number. A check is made to determine if the <i>SID</i> is ambiguous (e. g., r1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (e. g., r1.0 or r1.1.0 are invalid because neither case can exist as a valid delta number). If the <i>SID</i> is valid and not ambiguous, a check is made to determine if it actually exists. |
| -mname | The argument value <i>name</i> is compared with the SCCS %M% keyword in <i>file</i> .                                                                                                                                                                                                                                                                                                                                                                                      |
| -ytype | The argument value <i>type</i> is compared with the SCCS %Y% keyword in <i>file</i> .                                                                                                                                                                                                                                                                                                                                                                                      |

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

- bit 0 = missing file argument;
- bit 1 = unknown or duplicate keyletter argument;
- bit 2 = corrupted SCCS file;
- bit 3 = cannot open file or file not SCCS;
- bit 4 = *SID* is invalid or ambiguous;
- bit 5 = *SID* does not exist;
- bit 6 = %Y%, -y mismatch;
- bit 7 = %M%, -m mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned - a logical OR of the codes generated for each command line and file processed.

## **VAL(1)**

### **SEE ALSO**

**admin(1), delta(1), get(1), help(1), prs(1).**

### **DIAGNOSTICS**

Use *help(1)* for explanations.

### **BUGS**

*Val* can process up to 50 files on a single command line. Any number above 50 will produce a core dump.

**NAME**

vc — version control

**SYNOPSIS**

vc [-a] [-t] [-cchar] [-s] [keyword=value ... keyword=value]

**DESCRIPTION**

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the *-t* keyletter (see below). The default control character is colon (:), except as modified by the *-c* keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumeric; the first must be alphabetic. A value is any ASCII string that can be created with *ed(1)*; a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The *-a* keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

**Keyletter Arguments**

- a* Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in *vc* statements.
- t* All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.
- cchar* Specifies a control character to be used in place of :.
- s* Silences warning messages (not error) that are normally printed on the diagnostic output.

**Version Control Statements**

:dcl keyword[, ..., keyword]

Used to declare keywords. All keywords must be declared.

:asg keyword=value

Used to assign values to keywords. An *asg* statement overrides the assignment for the corresponding keyword on the *vc* command line and all previous *asg*'s for that keyword. Keywords declared, but not assigned values have null values.

:if condition

:

:end

Used to skip lines of the standard input. If the condition is true all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.

The syntax of a condition is:

```

<cond> ::= ["not"] <or>
<or> ::= <and> | <and> "!" <or>
<and> ::= <exp> | <exp> "&" <and>
<exp> ::= "(" <or> ")" | <value> <op> <value>
<op> ::= "=" | "!=" | "<" | ">"
<value> ::= <arbitrary ASCII string> | <numeric string>

```

The available operators and their meanings are:

|     |                                                                                                              |
|-----|--------------------------------------------------------------------------------------------------------------|
| =   | equal                                                                                                        |
| !=  | not equal                                                                                                    |
| &   | and                                                                                                          |
|     | or                                                                                                           |
| >   | greater than                                                                                                 |
| <   | less than                                                                                                    |
| ( ) | used for logical groupings                                                                                   |
| not | may only occur immediately after the <i>if</i> , and when present, inverts the value of the entire condition |

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

```

= != > < all of equal precedence
&
|

```

Parentheses may be used to alter the order of precedence.

Values must be separated from operators or parentheses by at least one blank or tab.

:::text

Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the -a keyletter.

:on

:off

Turn on or off keyword replacement on all lines.

:ctl char

Change the control character to char.

:msg message

Prints the given message on the diagnostic output.



**:err message**

Prints the given message followed by:

**ERROR: err statement on line ... (915)**

on the diagnostic output. *Vc* halts execution, and returns an exit code of 1.

**SEE ALSO**

*ed(1)*, *help(1)*.

**DIAGNOSTICS**

Use *help(1)* for explanations.

**EXIT CODES**

0 - normal

1 - any error

# VI(1)

## NAME

**vi** — screen-oriented (visual) display editor based on **ex**

## SYNOPSIS

```
vi [-t tag] [-r file] [-l] [-wn] [-x] [-R] [+command]
name ...
view [-t tag] [-r file] [-l] [-wn] [-x] [-R] [+command]
name ...
vedit [-t tag] [-r file] [-l] [-wn] [-x] [-R] [+com-
mand] name ...
```

## DESCRIPTION

**Vi** (visual) is a display-oriented text editor based on an underlying line editor **ex(1)**. It is possible to use the command mode of **ex** from within **vi** and vice-versa.

When using **vi**, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file. The *Vi Quick Reference* card, the *Introduction to Display Editing with Vi* and the *Ex Reference Manual* provide full details on using **vi**.

## INVOCATION

The following invocation options are interpreted by **vi**:

- t tag** Edit the file containing the *tag* and position the editor at its definition.
- rfile** Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
- l** **LISP** mode; indents appropriately for lisp code, the **O** **I** and **ll** commands in **vi** and *open* are modified to have meaning for *lisp*.
- wn** Set the default window size to *n*. This is useful when using the editor over a slow speed line.
- x** Encryption mode; a key is prompted for allowing creation or editing of an encrypted file.
- R** Read only mode; the **readonly** flag is set, preventing accidental overwriting of the file.
- +command** The specified **ex** command is interpreted before editing begins.

The *name* argument indicates files to be edited.

The *view* invocation is the same as **vi** except that the **readonly** flag is set.

The *vedit* invocation is intended for beginners. The **report** flag is set to 1, and the **showmode** and **novice** flags are set. These defaults make it easier to get started learning the editor.

## "VI MODES"

- Command** Normal and initial mode. Other modes return to command mode upon completion. ESC (escape) is used to cancel a partial command.
- Input.** Entered by **a i A I o O c C s S R**. Arbitrary text may then be entered. Input mode is normally terminated with ESC character, or abnormally with interrupt.
- Last line** Reading input for **:** **/** **?** or **!**; terminate with CR to execute, interrupt to cancel.

## COMMAND SUMMARY

## Sample commands

|                   |                            |
|-------------------|----------------------------|
| ←   ↑ →           | arrow keys move the cursor |
| h j k l           | same as arrow keys         |
| ite <i>xt</i> ESC | insert text <i>abc</i>     |
| ew <i>new</i> ESC | change word to <i>new</i>  |
| ea <i>s</i> ESC   | pluralize word             |
| x                 | delete a character         |
| dw                | delete a word              |
| dd                | delete a line              |
| 3dd               | ... 3 lines                |
| u                 | undo previous change       |
| ZZ                | exit vi, saving changes    |
| :q!CR             | quit, discarding changes   |
| /textCR           | search for <i>text</i>     |
| ^U ^D             | scroll up or down          |
| :ex cmdCR         | any ex or ed command       |

## Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

|                    |                  |
|--------------------|------------------|
| line/column number | z G              |
| scroll amount      | ^D ^U            |
| repeat effect      | most of the rest |

## Interrupting, canceling

|     |                                   |
|-----|-----------------------------------|
| ESC | end insert or incomplete cmd      |
| ^?  | (delete or rubout) interrupts     |
| ^L  | reprint screen if ^? scrambles it |
| ^R  | reprint screen if ^L is → key     |

## File manipulation

|                     |                                |
|---------------------|--------------------------------|
| :wCR                | write back changes             |
| :qCR                | quit                           |
| :q!CR               | quit, discard changes          |
| :e <i>name</i> CR   | edit file <i>name</i>          |
| :e!CR               | reedit, discard changes        |
| :e + <i>name</i> CR | edit, starting at end          |
| :e + <i>n</i> CR    | edit starting at line <i>n</i> |
| :e #CR              | edit alternate file            |
|                     | synonym for :e #               |
| :w <i>name</i> CR   | write file <i>name</i>         |
| :w! <i>name</i> CR  | overwrite file <i>name</i>     |
| :shCR               | run shell, then return         |
| !: <i>cmd</i> CR    | run <i>cmd</i> , then return   |
| :nCR                | edit next file in arglist      |
| :n <i>args</i> CR   | specify new arglist            |
| ^G                  | show current file and line     |
| :ta <i>tag</i> CR   | to tag file entry <i>tag</i>   |
| ^]                  | :ta, following word is tag     |

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a CR.

## Positioning within file

|                |                                    |
|----------------|------------------------------------|
| <b>^F</b>      | forward screen                     |
| <b>^B</b>      | backward screen                    |
| <b>^D</b>      | scroll down half screen            |
| <b>^U</b>      | scroll up half screen              |
| <b>G</b>       | go to specified line (end default) |
| <b>/pat</b>    | next line matching <i>pat</i>      |
| <b>?pat</b>    | prev line matching <i>pat</i>      |
| <b>n</b>       | repeat last / or ?                 |
| <b>N</b>       | reverse last / or ?                |
| <b>/pat/+n</b> | noth line after <i>pat</i>         |
| <b>?pat?-n</b> | noth line before <i>pat</i>        |
| <b>  </b>      | next section/function              |
| <b>[[</b>      | previous section/function          |
| <b>(</b>       | beginning of sentence              |
| <b>)</b>       | end of sentence                    |
| <b>{</b>       | beginning of paragraph             |
| <b>}</b>       | end of paragraph                   |
| <b>%</b>       | find matching ( ) { or }           |

## Adjusting the screen

|                  |                               |
|------------------|-------------------------------|
| <b>^L</b>        | clear and redraw              |
| <b>^R</b>        | retype, eliminate @ lines     |
| <b>zCR</b>       | redraw, current at window top |
| <b>z-CR</b>      | ... at bottom                 |
| <b>z.CR</b>      | ... at center                 |
| <b>/pat/z-CR</b> | <i>pat</i> line at bottom     |
| <b>zn.CR</b>     | use <i>n</i> line window      |
| <b>^E</b>        | scroll window down 1 line     |
| <b>^Y</b>        | scroll window up 1 line       |

## Marking and returning

|            |                                            |
|------------|--------------------------------------------|
| <b>``</b>  | move cursor to previous context            |
| <b>...</b> | ... at first non-white in line             |
| <b>mx</b>  | mark current position with letter <i>x</i> |
| <b>'x</b>  | move cursor to mark <i>x</i>               |
| <b>^x</b>  | ... at first non-white in line             |

## Line positioning

|               |                                   |
|---------------|-----------------------------------|
| <b>H</b>      | top line on screen                |
| <b>L</b>      | last line on screen               |
| <b>M</b>      | middle line on screen             |
| <b>+</b>      | next line, at first non-white     |
| <b>-</b>      | previous line, at first non-white |
| <b>CR</b>     | return, same as +                 |
| <b>  or j</b> | next line, same column            |
| <b>  or k</b> | previous line, same column        |

## Character positioning

|                      |                                        |
|----------------------|----------------------------------------|
| <b>^</b>             | first non-white                        |
| <b>0</b>             | beginning of line                      |
| <b>\$</b>            | end of line                            |
| <b>h</b> or <b>→</b> | forward                                |
| <b>l</b> or <b>←</b> | backwards                              |
| <b>^H</b>            | same as <b>←</b>                       |
| space                | same as <b>→</b>                       |
| <b>fx</b>            | find <i>x</i> forward                  |
| <b>Fx</b>            | <b>f</b> backward                      |
| <b>tx</b>            | upto <i>x</i> forward                  |
| <b>Tx</b>            | back upto <i>x</i>                     |
| <b>;</b>             | repeat last <b>f F t</b> or <b>T</b>   |
| <b>,</b>             | inverse of <b>;</b>                    |
| <b> </b>             | to specified column                    |
| <b>%</b>             | find matching ( <b>(</b> ) or <b>)</b> |

## Words, sentences, paragraphs

|          |                      |
|----------|----------------------|
| <b>w</b> | word forward         |
| <b>b</b> | back word            |
| <b>e</b> | end of word          |
| <b>)</b> | to next sentence     |
| <b>}</b> | to next paragraph    |
| <b>(</b> | back sentence        |
| <b>{</b> | back paragraph       |
| <b>W</b> | blank delimited word |
| <b>B</b> | back W               |
| <b>E</b> | to end of W          |

## Commands for LISP Mode

|          |                              |
|----------|------------------------------|
| <b>)</b> | Forward s-expression         |
| <b>)</b> | ... but do not stop at atoms |
| <b>(</b> | Back s-expression            |
| <b>{</b> | ... but do not stop at atoms |

## Corrections during insert

|              |                                        |
|--------------|----------------------------------------|
| <b>^H</b>    | erase last character                   |
| <b>^W</b>    | erase last word                        |
| <b>erase</b> | your erase, same as <b>^H</b>          |
| <b>kill</b>  | your kill, erase input this line       |
| <b>\</b>     | quotes <b>^H</b> , your erase and kill |
| <b>ESC</b>   | ends insertion, back to command        |
| <b>^?</b>    | interrupt, terminates insert           |
| <b>^D</b>    | backtab over <i>autoindent</i>         |
| <b>↑^D</b>   | kill <i>autoindent</i> , save for next |
| <b>0^D</b>   | ... but at margin next also            |
| <b>^V</b>    | quote non-printing character           |

## Insert and replace

|                 |                                   |
|-----------------|-----------------------------------|
| <b>a</b>        | append after cursor               |
| <b>i</b>        | insert before cursor              |
| <b>A</b>        | append at end of line             |
| <b>I</b>        | insert before first non-blank     |
| <b>o</b>        | open line below                   |
| <b>O</b>        | open above                        |
| <b>rx</b>       | replace single char with <i>x</i> |
| <b>RtextESC</b> | replace characters                |

**Operators**

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since *w* moves over a word, *dw* deletes the word that would be moved over. Double the operator, e.g., *dd* to affect whole lines.

|             |                        |
|-------------|------------------------|
| <b>d</b>    | delete                 |
| <b>c</b>    | change                 |
| <b>y</b>    | yank lines to buffer   |
| <b>&lt;</b> | left shift             |
| <b>&gt;</b> | right shift            |
| <b>!</b>    | filter through command |
| <b>=</b>    | indent for LISP        |

**Miscellaneous Operations**

|          |                                    |
|----------|------------------------------------|
| <b>C</b> | change rest of line ( <b>c\$</b> ) |
| <b>D</b> | delete rest of line ( <b>d\$</b> ) |
| <b>s</b> | substitute chars ( <b>cl</b> )     |
| <b>S</b> | substitute lines ( <b>cc</b> )     |
| <b>J</b> | join lines                         |
| <b>x</b> | delete characters ( <b>dl</b> )    |
| <b>X</b> | ... before cursor ( <b>db</b> )    |
| <b>Y</b> | yank lines ( <b>yy</b> )           |

**Yank and Put**

Put inserts the text most recently deleted or yanked. However, if a buffer is named, the text in that buffer is put instead.

|            |                             |
|------------|-----------------------------|
| <b>p</b>   | put back text after cursor  |
| <b>P</b>   | put before cursor           |
| <b>"xp</b> | put from buffer <i>x</i>    |
| <b>"xy</b> | yank to buffer <i>x</i>     |
| <b>"xd</b> | delete into buffer <i>x</i> |

**Undo, Redo, Retrieve**

|            |                                   |
|------------|-----------------------------------|
| <b>u</b>   | undo last change                  |
| <b>U</b>   | restore current line              |
| <b>.</b>   | repeat last change                |
| <b>"dp</b> | retrieve <i>d</i> 'th last delete |

**AUTHOR**

*Vi* and *ex* were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

**SEE ALSO**

*ex*(1).  
"Screen Editor Tutorial (*vi*)."

**CAVEATS AND BUGS**

Software tabs using **^T** work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

There should be an interactive *help* facility and a tutorial suited for beginners.

## NAME

volcopy, labelit -- copy file systems with label checking

## SYNOPSIS

```
/etc/volcopy [options] fsname special1 volname1 special2 volname2
/etc/labelit special [fsname volume [-n]]
```

## DESCRIPTION

*Volcopy* makes a literal copy of the file system using a blocksize matched to the device. *Options* are:

- a invoke a verification sequence requiring a positive operator response instead of the standard 10-second delay before the copy is made
- s (default) invoke the DEL if wrong verification sequence.

Other *options* are used only with floppy disks:

- flopnum beginning floppy number for a restarted copy,
- buf use double buffered I/O.

If the file system is too large to fit on one floppy disk, *volcopy* will prompt for additional floppies. Labels of all floppy disks are checked. If *volcopy* is interrupted, it will ask if the user wants to quit or wants a shell. In the latter case, the user can perform other operations (e.g.: *labelit*) and return to *volcopy* by exiting the new shell.

The *fsname* argument represents the mounted name (e.g.: **root, u1**, etc.) of the file system being copied.

The *special* should be the physical disk section (e.g.: **/dev/rdisk0s7**).

The *volname* is the physical volume name (e.g.: **pk3, t0122**, etc.) and should match the external label sticker. Such label names are limited to six or fewer characters. *Volname* may be **-** to use the existing volume name.

*Special1* and *volname1* are the device and volume from which the copy of the file system is being extracted. *Special2* and *volname2* are the target device and volume.

*Fsname* and *volname* are recorded in the last 12 characters of the super block (**char fsname[6], volname[6]**).

*Labelit* can be used to provide initial labels for unmounted disk or floppy file systems. With the optional arguments omitted, *labelit* prints current label values. The **-n** option provides for initial labeling of new backup floppies only (this destroys previous contents).

## FILES

**/etc/log/filesave.log** a record of file systems/volumes copied

## SEE ALSO

sh(1), fs(4).

## WAIT(1)

### NAME

wait — await completion of process

### SYNOPSIS

wait

### DESCRIPTION

Wait until all processes started with `&` have completed, and report on abnormal terminations.

Because the `wait(2)` system call must be executed in the parent process, the shell itself executes `wait`, without creating a new process.

### SEE ALSO

`sh(1)`.

`wait(2)` in the Software Development System manual.

### BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.



**NAME** wall - write to all users

**SYNOPSIS**  
/etc/wall

**DESCRIPTION**  
*Wall* reads its standard input until an end-of-file. It then sends this message to all currently logged-in users preceded by:

Broadcast Message from ...

It is used to warn all users; typically prior to shutting down the system.

The sender must be super-user to override any protections the users may have invoked [see *msg(1)*].

**FILES**  
/dev/tty\*

**SEE ALSO**  
*msg(1)*, *write(1)*.

**DIAGNOSTICS**  
"Cannot send to ..." when the open on a user's tty file fails.

## WC(1)

### NAME

`wc` — word count

### SYNOPSIS

`wc` [ `-lwc` ] [ *names* ]

### DESCRIPTION

*Wc* counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options `l`, `w`, and `c` may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is `-lwc`.

When *names* are specified on the command line, they will be printed along with the counts.

**NAME** what - identify SCCS files

**SYNOPSIS**  
what [-s] files

**DESCRIPTION**

*What* searches the given files for all occurrences of the pattern that *get(1)* substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file *f.c* contains

```
char ident[] = "@(#)identification information";
```

and *f.c* is compiled to yield *f.o* and *a.out*, then the command

```
what f.c f.o a.out
```

will print

```
f.c: identification information
```

```
f.o: identification information
```

```
a.out: identification information
```

*What* is intended to be used in conjunction with the command *get(1)*, which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

```
-s Quit after finding the first occurrence of pattern in each file.
```

**SEE ALSO**  
*get(1)*, *help(1)*.

**DIAGNOSTICS**

Exit status is 0 if any matches are found, otherwise 1. Use *help(1)* for explanations.

**BUGS**

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

## WHO(1)

### NAME

who — who is on the system

### SYNOPSIS

who [ **-uTHlpdbrtasq**] [ file ]

who am i

who am I

### DESCRIPTION

*Who* can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX system user. It examines the */etc/utmp* file to obtain this information. If *file* is given, that file is examined. Usually, *file* will be */etc/wtmp*, which contains a history of all the logins since the file was last created.

*Who* with the **am i** or **am I** option identifies the invoking user.

Except for the default **-s** option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

- u** This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory */dev*. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in */etc/inittab* [see *inittab(4)*]. This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.
- T** This option is the same as the **-u** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a - appears if it is not. *Root* can write to all lines having a + or a - in the *state* field. If a bad line is encountered, a ? is printed.
- l** This option lists only those lines on which the system is waiting for someone to log in. The *name* field is *LOGIN* in such cases. Other fields are the same as for user entries except that the *state* field does not exist.
- H** This option will print column headings above the regular output.
- q** This is a quick *who*, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.
- p** This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in */etc/inittab*. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from */etc/inittab* that spawned this process. See *inittab(4)*.

- d This option displays all processes that have expired and not been respawnd by *init*. The *exit* field appears for dead processes and contains the termination and exit values [as returned by *wait*(2)], of the dead process. This can be useful in determining why a process terminated.
- b This option indicates the time and date of the last reboot.
- r This option indicates the current *run-level* of the *init* process.
- t This option indicates the last change to the system clock [via the *date*(1) command] by *root*. See *su*(1).
- a This option processes */etc/utmp* or the named *file* with all options turned on.
- s This option is the default and lists only the *name*, *line*, and *time* fields.

**FILES**

*/etc/utmp*  
*/etc/wtmp*  
*/etc/inittab*

**SEE ALSO**

*date*(1), *init*(1M), *login*(1), *mesg*(1), *su*(1),  
*wait*(2), *inittab*(4), *utmp*(4) in the Software Development System manual.

## WHODO(1M)

### NAME

whodo -- who is doing what

### SYNOPSIS

*/etc/whodo*

### DESCRIPTION

*Whodo* produces merged, reformatted, and dated output from the *who(1)* and *ps(1)* commands.

### FILES

*etc/passwd*

### SEE ALSO

*ps(1)*, *who(1)*.

**NAME**

write — write to another user

**SYNOPSIS**

**write** user [ line ]

**DESCRIPTION**

*Write* copies lines from your terminal to that of another user. When first called, it sends the message:

*Message from yourname (tty??) [ date ]...*

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n". At that point *write* writes EOT on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., *tty00*); otherwise, the first writable instance of the user found in */etc/utmp* is assumed and the following message posted:

*user* is logged on more than one place.

You are connected to "*terminal*".

Other locations are:

*terminal*

Permission to write may be denied or granted by use of the *mesg(1)* command. Writing to others is normally allowed by default. Certain commands, in particular *nroff(1)* and *pr(1)* disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write-inhibited terminal.

If the character **!** is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal [i.e., **(o)** for "over"] so that the other person knows when to reply. The signal **(oo)** (for "over and out") is suggested when conversation is to be terminated.

**FILES**

*/etc/utmp* to find user

*/bin/sh* to execute !

**SEE ALSO**

*mail(1)*, *mesg(1)*, *pr(1)*, *sh(1)*, *who(1)*.

"Nroff and Troff User Manual" in the Text Preparation System manual.

**DIAGNOSTICS**

"*user is not logged on*" if the person you are trying to *write* to is not logged on.

"*Permission denied*" if the person you are trying to *write* to denies that permission (with *mesg*).

"*Warning: cannot respond, set mesg -y*" if your terminal is set to *mesg n* and the recipient cannot respond to you.

"*Can no longer write to user*" if the recipient has denied permission (*mesg n*) after you had started writing.

# XARGS(1)

## NAME

`xargs` - construct argument list(s) and execute command

## SYNOPSIS

`xargs` [*flags*] [*command* [*initial-arguments* ] ]

## DESCRIPTION

*Xargs* combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

*Command*, which may be a shell file, is searched for, using one's \$PATH. If *command* is omitted, `/bin/echo` is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see `-i` flag). Flags `-i`, `-l`, and `-n` determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., `-l` vs. `-n`), the last flag has precedence. *Flag* values are:

`-l`*number*

*Command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option `-x` is forced.

`-i`*replstr*

Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option `-x` is also forced. {} is assumed for *replstr* if not specified.

`-n`*number*

Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option `-x` is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.



- t** Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.
- p** Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**-t**) is turned on to print the command instance to be executed, followed by a *?... prompt*. A reply of *y* (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.
- x** Causes *xargs* to terminate if any argument list would be greater than *size* characters; **-x** is forced by the options **-i** and **-l**. When neither of the options **-i**, **-l**, or **-n** are coded, the total length of all arguments must be within the *size* limit.
- ssize** The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **-s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.
- eofstr** *Eofstr* is taken as the logical end-of-file string. Underbar (  ) is assumed for the logical EOF string if **-e** is not coded. The value **-e** with no *eofstr* coded turns off the logical EOF string capability (underbar is taken literally). *Xargs* reads standard input until either end-of-file or the logical EOF string is encountered.

*Xargs* will terminate if either it receives a return code of **-1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly exit [see *sh(1)*] with an appropriate value to avoid accidentally returning with **-1**.

#### EXAMPLES

The following will move all files from directory \$1 to directory \$2, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{ } $2/{ }
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*.

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

1. `ls | xargs -p -l ar r arch`
2. `ls | xargs -p -l | xargs ar r arch`

The following will execute *diff(1)* with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

SEE ALSO  
*sh(1)*.

DIAGNOSTICS  
Self-explanatory.

# YACC(1)

## NAME

yacc — yet another compiler-compiler

## SYNOPSIS

yacc [ -vdl ] grammar

## DESCRIPTION

*Yacc* converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, *y.tab.c*, must be compiled by the C compiler to produce a program *yparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex(1)* is useful for creating lexical analyzers usable by *yacc*.

If the *-v* flag is given, the file *y.output* is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the *-d* flag is used, the file *y.tab.b* is generated with the *#define* statements that associate the *yacc*-assigned "token codes" with the user-declared "token names". This allows source files other than *y.tab.c* to access the token codes.

If the *-l* flag is given, the code produced in *y.tab.c* will *not* contain any *#line* constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in *y.tab.c* under conditional compilation control. By default, this code is not included when *y.tab.c* is compiled. However, when *yacc*'s *-t* option is used, this debugging code will be compiled by default. Independent of whether the *-t* option was used, the runtime debugging code is under the control of *YYDEBUG*, a pre-processor symbol. If *YYDEBUG* has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

## FILES

*y.output*  
*y.tab.c*  
*y.tab.h* defines for token names  
*yacc.tmp*,  
*yacc.debug*, *yacc.acts* temporary files  
*/usr/lib/yaccparser* prototype for C programs

## SEE ALSO

*lex(1)*,  
*malloc(3X)* and "YACC — Yet Another Compiler Compiler" in the Software Development System manual.

## DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the *y.output* file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

## BUGS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.

## TABLE OF CONTENTS OF SPECIAL FILES

### 7. SPECIAL FILES

|              |                                |
|--------------|--------------------------------|
| intro.....   | introduction to special files  |
| console..... | screen driver interface        |
| err.....     | error-logging interface        |
| fl.....      | floppy disk drive              |
| io_op.....   | perform 286 I/O operations     |
| lp.....      | line printer                   |
| mem.....     | core memory                    |
| null.....    | the null file                  |
| pf.....      | operating system profiler      |
| sio.....     | serial interface               |
| termio.....  | general terminal interface     |
| tty.....     | controlling terminal interface |
| wd.....      | hard disk interface            |



**NAME**

intro — introduction to special files

**DESCRIPTION**

This section describes various special files that refer to specific hardware peripherals and System V/AT system device drivers. The names of the entries are generally derived from names for the hardware, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding System V/AT system device driver are discussed where applicable.

Disk device file names are in the following format:

`/dev/{r}dsk/{c#}d{#s#}`

where `r` indicates a raw interface to the disk, the `c#` indicates the controller number, and `#s#` indicates the drive and section numbers.

# CONSOLE(7)

## NAME

console — screen driver interface

## DESCRIPTION

The console driver controls the character attribute and screen mode settings for the system console. Both color and monochrome graphics boards are supported. Virtual console definitions are located in the /dev file and may contain alternate settings that the user may define. To access the virtual consoles from your system monitor, press the SYS REQ key on your keyboard, or use the ALT-F1 through ALT-F4 keys to access Screen-1 through Screen-4 directly. Each keypress gives you a new login prompt, and the fourth keypress returns you to your original login screen..

Standard ANSI termcap entries are supported along with some additional settings. These settings are listed as follows;

### Character attribute settings

Esc[0m      Switch to normal characters (white on black)  
Esc[1m      Boldface (brighten foreground color)  
Esc[2x;ym    Sets color graphic mode:  
              320 x 200, x = foreground and y = background  
              640 x 200, y = foreground and x = background  
Esc[3;0m    Switch from blink mode to bright background mode  
Esc[3;1m    Switch from bright background mode to blink mode  
Esc[4m      Underscore (monochrome only)  
Esc[5m      Blink characters or brighten background color  
Esc[7m      Reverse video (white on black)  
Esc[8m      Invisible characters  
Esc[10m     Set primary font to normal ASCII  
Esc[11m     Set first alternate font  
Esc[12m     Set second alternate font  
Esc[30m     Black foreground

### Screen mode settings

Esc[h        Switch to default mode (CMOS setting)  
Esc[0h      Switch to 40 x 25 black & white  
Esc[2h      Switch to 80 x 25 black & white  
Esc[3h      Switch to 80 x 25 color  
Esc[4h      Switch to 320 x 200 color  
Esc[5h      Switch to 320 x 200 black & white  
Esc[6h      Switch to 640 x 200 black & white

## MAJOR, MINOR DEVICE NUMBERS

| Maj., min | Special filename | Semantics         |
|-----------|------------------|-------------------|
| 0, 0      | /dev/console     | console           |
| 0, 1      | /dev/cons1       | virtual console 1 |
| 0, 2      | /dev/cons2       | virtual console 2 |
| 0, 3      | /dev/cons3       | virtual console 3 |

**NAME**

*err* — error-logging interface

**DESCRIPTION**

Minor device 0 of the *err* driver is the interface between a process and the system's error-record collection routines. The driver may be opened only for reading by a single process with super-user permissions. Each read causes an entire error record to be retrieved; the record is truncated if the readrequest is for less than the record's length.

**FILES**

*/dev/error* special file

**SEE ALSO**

*errdemon(1M)*

## FL(7)

### NAME

fl—floppy disk drive

### DESCRIPTION

The floppy driver provides an interface to popular disk drives compatible with a 512 byte sector disk format. The floppy driver lists minor devices as follows:

| name      | minor | drive | tpi | sectors | capacity (KB) | sides | density |
|-----------|-------|-------|-----|---------|---------------|-------|---------|
| fl        | 70    | 0     | 96  | 15      | 1200          | 2     | high    |
| fd096     | 70    | 0     | 96  | 15      | 1200          | 2     | high    |
| fd096ds15 | 70    | 0     | 96  | 15      | 1200          | 2     | high    |
| 0s24      | 70    | 0     | 96  | 15      | 1200          | 2     | high    |
| *0s25     | 198   | 0     | 96  | 15      | 1200          | 2     | high    |
| fd048     | 23    | 0     | 48  | 9       | 360           | 2     | low     |
| fd048ds9  | 23    | 0     | 48  | 9       | 360           | 2     | low     |
| fd096ds9  | 87    | 0     | 96  | 9       | 720           | 2     | high    |
| fd148     | 91    | 1     | 48  | 9       | 360           | 2     | low     |
| fd148ds9  | 91    | 1     | 48  | 9       | 360           | 2     | low     |
| fd196     | 78    | 1     | 96  | 15      | 1200          | 2     | high    |
| fd196ds15 | 78    | 1     | 96  | 15      | 1200          | 2     | high    |

\* = one cylinder offset (boot floppies and installit floppies use this device)

For block device access, the major device number is 1 and its special file is located in */dev/dsk*.

For raw device access, the major device number is 6 and its special file is located in */dev/rdsk*.

For example, to make a node for raw access to a high-density double-sided 1.2 MB floppy in the primary floppy drive, type "*mknod /dev/rdsk/fd c 6 70*". See *mknod(1)* for details of making nodes.

### MAJOR, MINOR DEVICE NUMBERS

| Maj., min. | Special filename       | Semantics                              |
|------------|------------------------|----------------------------------------|
| 1, 70      | <i>/dev/dsk/fd</i>     | high (1.2MB) #0                        |
| 1, 198     | <i>/dev/dsk/0s25</i>   | high (1.2MB) #0<br>(cylinder 1 onward) |
| 1, 23      | <i>/dev/dsk/fd048</i>  | low (360KB) #0                         |
| 1, 78      | <i>/dev/dsk/fd196</i>  | high (1.2MB) #1                        |
| 1, 91      | <i>/dev/dsk/fd148</i>  | low (360KB) #1                         |
| 6, 70      | <i>/dev/rdsk/fd</i>    | raw high (1.2MB) #0                    |
| 6, 198     | <i>/dev/rdsk/0s25</i>  | raw high (1.2MB) #0                    |
| 6, 23      | <i>/dev/rdsk/fd048</i> | raw low (360KB) #0                     |
| 6, 78      | <i>dev/rdsk/fd196</i>  | raw high (1.2MB) #1                    |
| 6, 91      | <i>dev/dsk/fd148</i>   | raw low (360KB) #1                     |



**NAME**

io\_op - perform 286 I/O operations

**SYNOPSIS**

```
#include <sys/io_op.h>
```

```
ioctl (fd, io_op, code);
```

```
int fd, code;
```

```
struct {
```

```
 unsigned int io_port; /* Port number */
```

```
 unsigned int io_word; /* Word data stored here */
```

```
 unsigned char io_byte; /* Byte data in here */
```

```
} *io_op;
```

```
#define IOCIOP_RB ((T<<8) | 0) /* Read a byte */
```

```
#define IOCIOP_RW ((T<<8) | 1) /* Read a word */
```

```
#define IOCIOP_WB ((T<<8) | 2) /* Write a byte */
```

```
#define IOCIOP_WW ((T<<8) | 3) /* Write a word */
```

**DESCRIPTION**

The io\_op interface allows user programs to perform 286 I/O operations, through an ioctl(2) call on /dev/mem. Code denotes whether a byte or a word is to be written or read. Io\_port contains the port number to be used, which must be in the range 0-0x3ff. Io\_word is the data to be written or read, for word operations. Io\_byte is the data to be written or read, for byte operations.

## LP(7)

### NAME

lp — parallel line printer

### DESCRIPTION

*Lp* provides the interface to any of the standard parallel line printers. When it is opened or closed, a suitable number of page ejects is generated. Bytes written are printed.

An internal parameter within the driver determines whether or not the device is treated as having a 96- or 64-character set. In half-ASCII mode, lowercase letters are turned into uppercase, and certain characters are escaped according to the following table:

|   |   |
|---|---|
| { | ← |
| } | → |
| - |   |
|   | + |
| - | ^ |

The driver correctly interprets carriage returns, backspaces, tabs, and form-feeds. A new-line that extends over the end of a page is turned into a form-feed. The default line length is 132 characters, indent is 0 characters and lines per page is 66. Lines longer than the line length minus the indent (i.e., 128 characters, using the above defaults) are truncated.

These defaults can be overridden with the *lpset(1)* command. *lpset(1)* can also be used to set transparent mode which disables interpretation of characters and suppresses page ejection whenever the lp device is opened or closed. *lpget(1)* retrieves the current settings.

Two *ioctl(2)* system calls are available:

```
#include <sys/lprio.h>
ioctl(fdles, command, arg)
struct lprio *arg;
```

The *commands* are:

- LPRGET** Get the current indent, columns per line, lines per page, and mode and store in the *lprio* structure referenced by *arg*.
- LPRSET** Set the current indent, columns per line, lines per page, and mode from the structure referenced by *arg*.

Thus, indent, page width, page length, and mode can be set with an external program. Two utilities can be used for this capability:

```
lpget device
lpset device indentation columns lines [transparency]
```

### MAJOR, MINOR DEVICE NUMBERS

| Maj., min. | Special filename | Semantics                                 |
|------------|------------------|-------------------------------------------|
| 7, 0       | /dev/lp 0        | parallel port on monochrome adaptor board |
| 7, 1       | /dev/lp 1        | parallel port 1 or parallel/serial card   |
| 7, 2       | /dev/lp 2        | parallel port 2 or parallel/serial card   |

### SEE ALSO

*ioctl(2)*, *lpset(1)*, *lpget(1)*.

**NAME**

mem,kmem—core memory

**DESCRIPTION**

Mem is a special file that is an image of the core memory of computer. It may be used, for example, to examine, and even to patch the system.

Byte addresses in mem are interpreted as memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file kmem is the same as mem except that kernel virtual memory rather than physical memory is accessed.

**FILES**

/dev/mem  
/dev/kmem

## **NULL(7)**

### **NAME**

null – the null file

### **DESCRIPTION**

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

### **FILES**

/dev/null

**NAME**

*prf* — operating system profiler

**DESCRIPTION**

The file *prf* provides access to activity information in the operating system. Writing the file loads the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file *prf* is a pseudo-device with no associated hardware.

**FILES**

*/dev/prf*

**SEE ALSO**

*config(1M)*, *profiler(1M)*.

**This page intentionally left blank.**

**NAME**

sio — serial interface

**SYNOPSIS**

sio

**DESCRIPTION**

Each line attached to the serial adaptor card behaves as described in `termio(7)`. Input and output for each line may independently be set to run at 110, 150, 300, 600, 1200, 2400, 4800, and 9600 baud. Only 7- and 8-bit character sizes are supported. Output speed is always input speed.

There are 3 device names for each port: `/dev/tty0`, `/dev/ttyM0`, and `/dev/ttyM0`, all refer to DOS COM1; and similarly `/dev/tty1`, `/dev/ttyM1`, and `/dev/ttyM1` all refer to COM2.

`/dev/tty0` is to be used for 'cu', 'kermit', and 'uucp', serial printers, and gettys for directly connected terminals.

`/dev/ttyM0` is normally to be used for gettys on an auto-answer modem.

`/dev/ttyM0` is to be used when simultaneously allowing gettys and 'cu' or 'uucp' dial-out usage on an auto-dial/auto-answer modem or direct connection to another computer.

`/dev/tty0` totally ignores all four modem input lines. It asserts DTR when opened, and, if HUPCL is set — see `termio(7)`, drops it when closed.

`/dev/ttyM0` waits for DCD when opened, but if `/dev/tty0` is in use, `/dev/ttyM0` waits until it is not. If `/dev/ttyM0` is open and in use, `/dev/tty0` will simply return an I/O error on open. The first mechanism allows 'uucp' and 'cu' to dial out on a modem (using `tty0`), while the getty(1) blocks in its `open(2)` waiting for the modem to assert DCD (on `tyM0`). The second mechanism disallows 'cu' or 'uucp' (on `ty0`) if someone is already logged in (on `tyM0`). These dial programs also lock out each other using `/usr/spool/uucp/LCK...` files.

Discover the type of interface (DTE or DCE) for the equipment you are trying to connect, and use a cable that will interface from that to the DTE type interface on nearly all PC serial ports. Modems and "Null Modems" are DCE (Data COMMUNICATIONS Equipment), Terminals, PC serial ports, printers and computer ports are DTE (Data TERMINAL Equipment). With a "straight through" cable, one end must look like DTE, and the other must be DCE. A "Null Modem" presents a DCE interface to two DTE devices, but cannot supply RI (Ring Indicator) of course. The standard 8250/16450-based serial port adapters have a serious design flaw: they do not place termination resistors on modem input lines. Internal Modem cards are especially notorious for this. Some cards have a problem where the modem input lines between UART & phone interface will oscillate, thus continually interrupting the operating system, effectively halting the system. `/dev/tty0` and `/dev/tty1` do not have this problem, as they ignore modem interrupts; but `/dev/ttyM` and `/dev/ttyM` do. For best results, use the following wiring scheme for building serial cables:

| Terminal/printer: |        | Modem: |        | Another PC/AT: |              |
|-------------------|--------|--------|--------|----------------|--------------|
| From              | To     | From   | To     | From           | To           |
| 9-pin             | 25-pin | 9-pin  | 25-pin | 9-pin          | remote 9-pin |
| 1,6               | 20     | 1      | 8      | 1,6            | 4            |
| 2                 | 2      | 2      | 3      | 2              | 3            |
| 3                 | 3      | 3      | 2      | 3              | 2            |
| 4                 | 6,8    | 4      | 20     | 4              | 1,6          |
| 5                 | 7      | 5      | 7      | 5              | 5            |
| 7                 | 5      | 6      | 6      | 7              | 8            |
| 8                 | 4      | 7      | 4      | 8              | 7            |
|                   |        | 8      | 5      |                |              |
|                   |        | 9      | 22     |                |              |

Refer to the following pin description while deciphering the above:

| PC/AT 9-pin(DTE): | RS-232 25-pin DTE; | RS-232 25-pin DCE; |
|-------------------|--------------------|--------------------|
| shell/shield      | 1 frame ground     | 1 frame ground     |
| 3 TD out          | 2 TD out           | 2 TD in            |
| 2 RD in           | 3 RD in            | 3 RD out           |
| 7 RTS out         | 4 RTS out          | 4 RTS in           |
| 8 CTS in          | 5 CTS in           | 5 CTS out          |
| 6 DSR in          | 6 DSR in           | 6 DSR out          |
| 5 signal GND      | 7 signal GND       | 7 signal GND       |
| 1 DCD in          | 8 DCD in           | 8 DCD out          |
| 4 DTR out         | 20 DTR out         | 20 DTR in          |
| 9 RI in           | 22 RI in           | 22 RI out          |

Connectors on the card are typically male and numbered (looking into back):

|         |    |         |   |                |
|---------|----|---------|---|----------------|
| 1 ..... | 13 | 1 ..... | 5 | Type DB-25/9 P |
| 14..... | 25 | 6.....  | 9 |                |

## HOW TO INSTALL A MODEM

To connect a modem to an asynchronous communications line, use the following procedures. The Hayes smart modem 1200 is used as an example of how to install a modem. An External type of modem is recommended over an internal card (External Modems can be moved to a smart serial board) as the control over line termination and loop back is usually limited to a few jumpers on the card.

1. Set your modem switches for auto answer or originate only (Switch #5).

The Hayes settings which are important for proper operation are:

| Hayes switch # | auto answer | originate  | new semantics                                                 |
|----------------|-------------|------------|---------------------------------------------------------------|
| 1              | up          | up         | monitor DTR & D2 (on 2400)<br>(drop connection if CLOCAL)     |
| 2              | up          | up         | display result codes as words<br>(for use with dial programs) |
| 3              | up<br>down  | up<br>down | 'quiet' mode: no cmd response<br>OK, CONNECT, etc.            |
| 4              | up          | up         | commands echoed                                               |
| 5              | up          | down       | auto answer                                                   |
| 6              | up          | up         | monitor CD & CI (on 2400)<br>(allows use of ttyM0)            |

2. Patch the operating system if necessary with the *typatch(1)* shell script and power down the PC-AT with Cntrl-Alt-Del.
3. Install serial interface and/or multipoint cards as desired with interrupt and address jumpers set as patched in (2).
4. Connect the modem cable between the card's connector and that on the back of the Hayes 1200.



5. Power up the PC-AT.
6. /dev/tty0 is minor device 0, /dev/tty0 is minor device 128, /dev/ttyM0 is minor device 192. Add 1 for tty1, 2 for tty2 and so on. If you are planning to use either the special files /dev/tty0, /dev/ttyM0 (or /dev/tty1, /dev/ttyM1) which are already on your system.

The major, minor device numbers and the semantics are as follows:

Major=5, #=Minor(0-32): M=128+64+# m=128+# for /dev/tty#

|       |            |                         |
|-------|------------|-------------------------|
| 5,0   | /dev/tty0  | originate only mode     |
| 5,128 | /dev/tty0  | answer only mode        |
| 5,192 | /dev/ttyM0 | answer/originate mode   |
|       |            | (recommended for getty) |
| 5,1   | /dev/tty1  | originate only mode     |
| 5,129 | /dev/ttyM1 | answer only mode        |
| 5,193 | /dev/ttyM1 | answer/originate mode   |

7. Set up an entry in /etc/inittab using the editor, as follows:

To allow both answer and originate modes:

```
t0:23:respawn:/etc/getty ttyM0 1200
```

To allow only cu/uucp dial-outs (originate mode)

```
t0:23:off:/etc/getty tty0 1200
```

After leaving the editor, type:

```
telinit q
```

to tell init(1M) to reread (query) the updated /etc/inittab file. This can be verified by typing "ps -ef" (which should show /etc/getty running on the tty in the etc/getty line of inittab). If the baud rate is changed in inittab after getty was already spawned for the old speed, you must kill(1) it so that the new getty will use the new inittab value.

8. As of the 2.2 release of System V/AT, cu(1) and uucp(1) have been upgraded. A public domain version of the standard System V "dial" subroutine replaces the older version.

A new dialing information file, called /usr/lib/uucp/dialinfo, now supports operation of different types of modems. The format is documented in the Software Development System manual, see dialinfo(4). As a consequence of the new dial support, you will need to change the L-devices file. The third field now designates the modem type. Here is an example L-devices entry for a Hayes compatible modem

operated at 1200 baud on tty0:

L-devices (single line string):

```
ACU tty0 Hayes 1200
```

Any direct connection should specify "direct" for the modem type:

```
DIR tty1 direct 9600
```

The syntax "cu systemname" is also supported in the new software so that cu(1) will automatically call out according to the telephone number represented in the L.sys entry for systemname. You must enable result responses in english (Hayes SW3 down) for this to work.

9. Because automatic dialing is now supported you may wish to change your L.sys entries to operate with the ACU device rather than as hardwired connections. Refer to the discussion of the system file in Chapter 12, "Uucp Administration." An example follows:

L.sys (single line string):

```
sysname Any ACU 1200 xxxxxxxx ogin: nuucp ssword: yyyyy
```

where sysname is replaced with the name of the system being called, xxxxxxxx is replaced with the phone number of the system, and yyyyy is replaced with the uucp password for that system.

10. In order to support the older version of uucp, which was distributed with the Microport 1.3 release, use the following format for L.sys:

```
sysname Any tty0 1200 tty0 "" "" "" ATDT\xxxxxxx\ ECT @ ogin: nuucp
ssword: yyyyy
```

## NAME

termio — general terminal interface

## DESCRIPTION

All of the asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files; they are opened by *getty* and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(2). A process can break this association by changing its process group using *setpgrp*(2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, the character # erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character @ kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The erase and kill characters may be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

- INTR (Rubout or ASCII DEL) generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).
- QUIT (Control-| or ASCII FS) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called *core*) will be created in the current working directory.
- SWTCH ASCII SM NUL is used by the job control facility, *shl*, to change the current layer to the control layer. (Not on PDP-11).

## TERMIO(7)

- ERASE** (#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.
- KILL** (@) deletes the entire line, as delimited by a NL, EOF, or EOL character.
- EOF** (Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.
- NL** (ASCII LF) is the normal line delimiter. It can not be changed or escaped.
- EOL** (ASCII NUL) is an additional line delimiter, like NL. It is not normally used.
- STOP** (Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.
- START** (Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, SWTCH, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a *hang-up* signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hang-up signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Several *ioctl(2)* system calls apply to terminal files. The primary calls use the following structure, defined in `<termio.h>`:

```
#define NCC 8
struct termio {
 unsigned short c_iflag; /* input modes */
 unsigned short c_oflag; /* output modes */
 unsigned short c_cflag; /* control modes */
 unsigned short c_lflag; /* local modes */
 char c_line; /* line discipline */
 unsigned char c_cc[NCC]; /* control chars */
};
```

The special control characters are defined by the array `c_cc`. The relative positions and initial values for each function are as follows:

|   |          |     |
|---|----------|-----|
| 0 | VINTR    | DEL |
| 1 | VQUIT    | FS  |
| 2 | VERASE   | #   |
| 3 | VKILL    | @   |
| 4 | VEOF     | EOT |
| 5 | VEOL     | NUL |
| 6 | reserved |     |
| 7 | SWTCH    | NUL |

The `c_iflag` field describes the basic terminal input control:

|        |         |                                         |
|--------|---------|-----------------------------------------|
| IGNBRK | 0000001 | Ignore break condition.                 |
| BRKINT | 0000002 | Signal interrupt on break.              |
| IGNPAR | 0000004 | Ignore characters with parity errors.   |
| PARMRK | 0000010 | Mark parity errors.                     |
| INPCK  | 0000020 | Enable input parity check.              |
| ISTRIP | 0000040 | Strip character.                        |
| INLCR  | 0000100 | Map NL to CR on input.                  |
| IGNCR  | 0000200 | Ignore CR.                              |
| ICRNL  | 0000400 | Map CR to NL on input.                  |
| IUCLC  | 0001000 | Map uppercase to lowercase on input.    |
| IXON   | 0002000 | Enable start/stop output control.       |
| IXANY  | 0004000 | Enable any character to restart output. |
| IXOFF  | 0010000 | Enable start/stop input control.        |

If `IGNBRK` is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if `BRKINT` is set, the break condition will generate an interrupt signal and flush both the input and output queues. If `IGNPAR` is set, characters with other framing and parity errors are ignored.

If `PARMRK` is set, a character with a framing or parity error which is not ignored is read as the three-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if `ISTRIP` is not set, a valid character of 0377 is read as 0377, 0377. If `PARMRK` is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If `INPCK` is set, input parity checking is enabled. If `INPCK` is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If `ISTRIP` is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If `INLCR` is set, a received NL character is translated into a CR character. If `IGNCR` is set, a received CR character is ignored (not read). Otherwise if `ICRNL` is set, a received CR character is translated into a NL character.

If `IUCLC` is set, a received uppercase alphabetic character is translated into the corresponding lowercase character.

If `IXON` is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If `IXANY` is set, any input character, will restart output which has been suspended.

If `IXOFF` is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

## TERMIO(7)

The initial input control value is all-bits-clear.

The *c\_oflag* field specifies the system treatment of output:

|        |         |                                   |
|--------|---------|-----------------------------------|
| OPOST  | 0000001 | Postprocess output.               |
| OLCUC  | 0000002 | Map lowercase to upper on output. |
| ONLCR  | 0000004 | Map NL to CR-NL on output.        |
| OCRNL  | 0000010 | Map CR to NL on output.           |
| ONOCR  | 0000020 | No CR output at column 0.         |
| ONLRET | 0000040 | NL performs CR function.          |
| OFILL  | 0000100 | Use fill characters for delay.    |
| OFDEL  | 0000200 | Fill is DEL, else NUL.            |
| NLDLY  | 0000400 | Select new-line delays:           |
| NL0    | 0       |                                   |
| NL1    | 0000400 |                                   |
| CRDLY  | 0003000 | Select carriage-return delays:    |
| CR0    | 0       |                                   |
| CR1    | 0001000 |                                   |
| CR2    | 0002000 |                                   |
| CR3    | 0003000 |                                   |
| TABDLY | 0014000 | Select horizontal-tab delays:     |
| TAB0   | 0       |                                   |
| TAB1   | 0004000 |                                   |
| TAB2   | 0010000 |                                   |
| TAB3   | 0014000 | Expand tabs to spaces.            |
| BSDLY  | 0020000 | Select backspace delays:          |
| BS0    | 0       |                                   |
| BS1    | 0020000 |                                   |
| VTDLY  | 0040000 | Select vertical-tab delays:       |
| VT0    | 0       |                                   |
| VT1    | 0040000 |                                   |
| FFDLY  | 0100000 | Select form-feed delays:          |
| FF0    | 0       |                                   |
| FF1    | 0100000 |                                   |

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2, four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c\_flag* field describes the hardware control of the terminal:

|        |         |                               |
|--------|---------|-------------------------------|
| CBAUD  | 0000017 | Baud rate:                    |
| B0     | 0       | Hang up                       |
| B50    | 0000001 | 50 baud                       |
| B75    | 0000002 | 75 baud                       |
| B110   | 0000003 | 110 baud                      |
| B134   | 0000004 | 134.5 baud                    |
| B150   | 0000005 | 150 baud                      |
| B200   | 0000006 | 200 baud                      |
| B300   | 0000007 | 300 baud                      |
| B600   | 0000010 | 600 baud                      |
| B1200  | 0000011 | 1200 baud                     |
| B1800  | 0000012 | 1800 baud                     |
| B2400  | 0000013 | 2400 baud                     |
| B4800  | 0000014 | 4800 baud                     |
| B9600  | 0000015 | 9600 baud                     |
| EXTA   | 0000016 | External A                    |
| EXTB   | 0000017 | External B                    |
| CSIZE  | 0000060 | Character size:               |
| CS5    | 0       | 5 bits                        |
| CS6    | 0000020 | 6 bits                        |
| CS7    | 0000040 | 7 bits                        |
| CS8    | 0000060 | 8 bits                        |
| CSTOPB | 0000100 | Send two stop bits, else one. |
| CREAD  | 0000200 | Enable receiver.              |
| PARENB | 0000400 | Parity enable.                |
| PARODD | 0001000 | Odd parity, else even.        |
| HUPCL  | 0002000 | Hang up on last close.        |
| CLOCAL | 0004000 | Local line, else dial-up.     |
| LOBLK  | 0010000 | Block layer output.           |

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stops bits are required.

## TERMO(7)

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise modem control is assumed.

If LOBLK is set, the output of a job control layer will be blocked when it is not the current layer. Otherwise the output generated by that layer will be multiplexed onto the current layer. (Not on PDP-11).

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The *cflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

|        |         |                                              |
|--------|---------|----------------------------------------------|
| ISIG   | 0000001 | Enable signals.                              |
| ICANON | 0000002 | Canonical input (erase and kill processing). |
| XCASE  | 0000004 | Canonical upper/lower presentation.          |
| ECHO   | 0000010 | Enable echo.                                 |
| ECHOE  | 0000020 | Echo erase character as BS-SP-BS.            |
| ECHOK  | 0000040 | Echo NL after kill character.                |
| ECHONL | 0000100 | Echo NL.                                     |
| NOFLSH | 0000200 | Disable flush after interrupt or quit.       |

If ISIG is set, each input character is checked against the special control characters INTR, SWTCH, and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g., 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters, respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an uppercase letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

| for: | use: |
|------|------|
| \    | !!   |
| ~    | ^    |
| {    | <    |
| }    | >    |
| \    | //   |



For example, A is input as `\a`, `\n` as `\\n`, and `\N` as `\\\n`.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit, switch, and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The primary `ioctl(2)` system calls have the form:

```
ioctl (files, command, arg)
struct termio *arg;
```

The commands using this form are:

|         |                                                                                                                                            |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------|
| TCGETA  | Get the parameters associated with the terminal and store in the <i>termio</i> structure referenced by <i>arg</i> .                        |
| TCSETA  | Set the parameters associated with the terminal from the structure referenced by <i>arg</i> . The change is immediate.                     |
| TCSETAW | Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output. |
| TCSETAF | Wait for the output to drain, then flush the input queue and set the new parameters.                                                       |

Additional `ioctl(2)` calls have the form:

```
ioctl (files, command, arg)
int arg;
```

The commands using this form are:

|        |                                                                                                                        |
|--------|------------------------------------------------------------------------------------------------------------------------|
| TCSBRK | Wait for the output to drain. If <i>arg</i> is 0, then send a break (zero bits for 0.25 seconds).                      |
| TCXONC | Start/stop control. If <i>arg</i> is 0, suspend output; if 1, restart suspended output.                                |
| TCFLSH | If <i>arg</i> is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues. |

#### FILES

`/dev/tty*`

#### SEE ALSO

`stty(1)`, `fork(2)`, `ioctl(2)`, `setpgrp(2)`, `signal(2)`.

## TTY(7)

### NAME

tty — controlling terminal interface

### DESCRIPTION

The file `/dev/tty` is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

### FILES

`/dev/tty`  
`/dev/tty*`

## NAME

wn— hard disk interface

## DESCRIPTION

The format for the disk files is described in intro(7). Files are accessed via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a raw interface (r) that provides for direct transmission between the disk and the user's read/write buffer.

## MAJOR, MINOR DEVICE NUMBERS

| Maj. min.            | Special filename                     | Semantics                                 |
|----------------------|--------------------------------------|-------------------------------------------|
| 0, 0                 | /dev/dsk/0s0                         | for drive #0: /root filesystem            |
| 0, 1                 | /dev/dsk/0s1                         | system swap area                          |
| 0, 2                 | /dev/dsk/0s2                         | /usr file system                          |
| 0, 3                 | /dev/dsk/0s3                         | /tmp file system                          |
| 0, 4                 | /dev/dsk/0s4                         | reserved                                  |
| 0, 5                 | /dev/dsk/0s5                         | DOS partition                             |
| 0, 6                 | /dev/dsk/0s6                         | partition number 1                        |
| 0, 7                 | /dev/dsk/0s7                         | partition number 2                        |
| 0, 8                 | /dev/dsk/0s8                         | partition number 3                        |
| 0, 9                 | /dev/dsk/0s9                         | partition number 4                        |
| 0, 10                | /dev/dsk/0s10                        | entire disk                               |
| 0, 11                | /dev/dsk/0s11                        | last track of active partition            |
| 0, 255               | /dev/dsk/0s255                       | last track of active partition            |
| 0, 20<br>to<br>0, 31 | /dev/dsk/1s0<br>/dev/dsk/1s1<br>etc. | for drive #1: same as minor 0 to 11 above |
| 4, 0                 | /dev/rdisk/0s0                       | raw interface                             |
| 4, 1                 | /dev/rdisk/0s1                       | to minor 0-31, above                      |
| etc.                 |                                      |                                           |

( )

( )

( )

**NOTES**





**NOTES**







**NOTES**



( )

—

( )

( )

**NOTES**

①

②

③

**NOTES**

( )

( )

( )

# NOTES

# NOTES



# NOTES

# NOTES