INTERACTIVE UNIX* System V/386
Release 3.2
User's/System Administrator's
Reference Manual

INTERACTIVE

product family

INTERACTIVE

A Kodak Company

First printing (March 1990)

Introduction

This manual describes the features of INTERACTIVE UNIX* System V/386 Release 3.2. It provides neither a general overview of the UNIX System nor details of the implementation of the system. Commands that constitute the basic software running on your computer are described.

The manual is divided into five sections:

- 1 – System Commands, System Maintenance Commands, and Application Programs
- 4 – File Formats†
- 5 – Miscellaneous Facilities†
- 7 – Special Files
- 8 – Additional System Administration Entries

Throughout this manual each reference of the form *name*(1M), *name*(4), *name*(5), *name*(7), or followed by a (1), (1C), (1G), or (1P) refers to an entry in this manual. The numbers following the command are intended for easy cross-reference. (Section 1 commands appropriate for use by programmers are located in the *INTERACTIVE Software Development System Guide and Programmer's Reference Manual.*) All other references to entries of the form *name*(N), where *N* is a number ((2), (3), (4), or (5)) possibly followed by a letter, refer to entry *name* in Section *N* of the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

Entries of the form *name*(*n*P) refer to entries that pertain to the operating system environment specified by U.S. Government standard *IEEE Std. 1003.1-1988.*

---

† Certain Section 4 and Section 5 entries from the *INTERACTIVE SDS Guide and Programmer's Reference Manual* that are applicable to system users as well as programmers have been repeated in this manual as well.

Each entry in the Commands section appears under a single name shown at the upper corners of its page(s). Entries are alphabetized, with the exception of the *intro*(1) entry, which is first. Entries may consist of more than one page. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its primary name, the name that appears at the upper corners of the page. An example of such an entry is *mount*(1M), which also describes the *umount* command. The secondary commands are listed directly below their associated primary command.

Section 1 (*System Commands, System Maintenance Commands, and Application Programs*) contains commands and programs that are:

- Used in administering a UNIX System.
- Invoked directly by the user or by command language procedures, as opposed to subroutines, which are called by the user's programs.

Commands generally reside in the directory **/bin** (for **bin**ary programs). In addition, some programs reside in **/usr/bin**. These directories are searched automatically by the command interpreter called the *shell*. The *shell* will search the path in your **.profile**. Make sure you have set this path in your **.profile** file. UNIX Systems running on your computer also have a directory called **/usr/lbin**, containing local commands.

The following sub-classes are in this section:

- 1 – General-purpose Commands
- 1C – Communications Commands
- 1G – Graphics Commands
- 1M – Maintenance Commands
- 1P – POSIX Commands

Each entry in the Commands section appears under a single name shown at the upper corners of its page(s).

Section 7 (*Special Files*) discusses the characteristics of system files that refer to input/output devices. The names in this section generally refer to device names for the hardware, rather than to the names of the special files themselves.

Section 8 (*System Administration*) discusses the Berkeley Sendmail Facility, the MS-DOS* (DOS) File System Switch, and the Kernel Debugger for the Intel* Architecture.

All entries are presented using the following format (though some of these headings might not appear in every entry):

- **NAME** gives the primary name (and secondary name(s), as the case may be) and briefly states its purpose.
- **SYNOPSIS** summarizes the usage of the program being described. A few explanatory conventions are used, particularly in Section 1M and the **SYNOPSIS**:
  - **Boldface** strings are literals and are to be typed just as they appear.
  - *Italic* strings usually represent substitutable argument prototypes and command names found elsewhere in the manual. (They are underlined in the on-line formatted versions of the entries.)
  - Square brackets [ ] around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file," it always refers to a *file* name.
  - Ellipses ... are used to show that the previous argument prototype may be repeated.
  - A final convention is used by the commands themselves. An argument beginning with a minus (−), plus (+), or an equal sign (=) is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with −, +, or =.
- **DESCRIPTION** discusses how to use these commands.
- **EXAMPLE(S)** gives example(s) of usage, where appropriate.
- **FILES** contains the file names that are referenced by the program.
- **EXIT CODES** discusses values set when the command terminates. The value set is available in the shell environment variable '?' (see *sh*(1)).
- **NOTES** gives information that may be helpful under the particular circumstances described.

- **SEE ALSO** offers pointers to related information.
- **DIAGNOSTICS** discusses the error messages that may be produced. Messages that are intended to be self-explanatory are not listed.
- **WARNINGS** discusses the limits or boundaries of the respective commands.
- **BUGS** lists known faults in software that have not been rectified. Occasionally, a suggested short-term remedy is also described.

Preceding Section 1 are a Table of Contents (listing both primary and secondary command entries) and a Permuted Index. Each line of the Table of Contents lists an abstract of a command. The Permuted Index is used by searching the middle column for a key word or phrase. The right column will then contain the name of the manual entry that contains the command. The left column contains additional useful information about the command.

Documentation References

Throughout this manual, the following full documentation titles will be referenced in shortened versions as follows:

Full Title	*Shortened Version*
INTERACTIVE Software Development System Guide and Programmer's Reference Manual	INTERACTIVE SDS Guide and Programmer's Reference Manual
INTERACTIVE UNIX System V/386 Release 3.2 User's/System Administrator's Reference Manual	INTERACTIVE UNIX System User's/System Administrator's Reference Manual
INTERACTIVE UNIX System V/386 Release 3.2 Operating System Guide	INTERACTIVE UNIX Operating System Guide
INTERACTIVE UNIX System V/386 Release 3.2 Guide for New Users	INTERACTIVE UNIX System Guide for New Users
INTERACTIVE UNIX System V/386 Release 3.2 User's Guide	User's Guide

How to Get Started

This discussion provides the basic information you need to get started on the UNIX System: how to log in and log out, how to communicate through your terminal, and how to run a program. (See the *User's Guide* for a more complete introduction to the system.)

Logging In

You must connect to the UNIX System from the console or a full-duplex ASCII terminal. You must also have a valid login ID, which may be obtained (together with instructions on how to access your UNIX System) from the administrator of your system. Common terminal speeds are 120, 240, 480, and 960 characters per second (1200, 2400, 4800, and 9600 baud). Some UNIX Systems have different ways of accessing each available terminal speed, while other systems offer several speeds through a common access method. In the latter case, there is one "preferred" speed; if you access it from a terminal set to a different speed, you will be greeted by a string of meaningless characters (the **login:** message at the wrong speed). Keep pressing the break, interrupt, or attention key until the **login:** message appears.

Most terminals have a speed switch that should be set to the appropriate speed and a half-/full-duplex switch that should be set to full-duplex. When a connection has been established, the system types **login:**. You respond by typing your login ID followed by the return key. If you have a password, the system asks for it but will not print, or echo, it on the terminal. After you have logged in, the return, new-line, and line-feed keys all have equivalent meanings.

Make sure you type your login name in lowercase letters. Typing uppercase letters causes the UNIX System to assume that your terminal can generate only uppercase letters and will treat all letters as uppercase for the remainder of your login session.

When you log in, a message-of-the-day may greet you before you receive your prompt. For more information, consult *login*(1), which discusses the login sequence in more detail, and *stty*(1), which tells you how to describe your terminal to the system. The entry *profile*(4) in this manual explains how to accomplish this last task automatically every time you log in.

Logging Out

There are two ways to log out:

- If you have dialed in, you can simply hang up the phone.
- You can log out by typing an end-of-file indication (ASCII **EOT** character, usually typed as **CTRL-D**) to the shell. The shell will terminate, and the **login:** message will appear again.

How to Communicate Through Your Terminal

When you type to the UNIX System, your individual characters are being gathered and temporarily saved. Although they are echoed back to you, these characters will not be given to a program until you type a return (or new-line) as described above in "Logging In."

UNIX System terminal input/output is full duplex. It has full read-ahead, which means that you can type at any time, even while a program is displaying information for you. Of course, if you type during output, your input characters will have output characters interspersed among them. In any case, whatever you type will be saved and interpreted in the correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded.

The character @ cancels all the characters typed before it on a line, effectively deleting the line. (@ is called the line kill character.) The character # erases the last character typed. Successive uses of # will erase characters back to, but not beyond, the beginning of the line; @ and # can be typed as themselves by preceding them with \ (thus, to erase a \, you need two #s). These default erase and line kill characters can be changed; see *stty*(1).

CTRL-S (also known as the ASCII **DC3** character) is typed by pressing the control key and the alphabetic **s** simultaneously and is used to stop output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. Output is resumed when a **CTRL-Q** (also known as **DC1**) is typed. Thus, if you had typed **cat yourfile** and the contents of **yourfile** were passing by on the screen more rapidly than you could read it, you would type **CTRL-S** to freeze the output. Typing **CTRL-Q** would allow the output to resume its rapid pace. The **CTRL-S** and **CTRL-Q** characters are not passed to any other program when used in this manner.

The ASCII **DEL** (a.k.a. rubout) character is not passed to programs but instead generates an *interrupt signal,* just like the break, interrupt, or attention signal. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you do not want. Programs, however, can arrange either to ignore this signal altogether or to be notified and take a specific action when it happens (instead of being terminated). The editor *ed*(1), for example, catches interrupts and stops what *it* is doing, instead of terminating, so an interrupt can be used to halt an editor printout without losing the file being edited.

Besides adapting to the speed of the terminal, the UNIX System tries to be intelligent as to whether you have a terminal with the "new-line" function, or whether it must be simulated with a "carriage-return" and "line-feed" pair. In the latter case, all *input* "carriage-return" characters are changed to "line-feed" characters (the standard line delimiter), and a "carriage-return" and "line-feed" pair is echoed to the terminal.

Tab characters are used freely in UNIX System source programs. If your terminal does not have the tab function, you can arrange to have tab characters changed into spaces during output, and echoed as spaces during input. Again, the *stty*(1) command will set or reset this mode. The system assumes that tabs are set every eight character positions. The *tabs*(1) command will set tab stops on your terminal, if that is possible.

How to Run a Program

When you have successfully logged in to the UNIX System, a program called the shell is communicating with your terminal. The shell reads each line you type, splits the line into a command name and its arguments, and executes the command. A command is simply an executable program. Normally, the shell looks first in your current directory (see "The Current Directory" below) for a program with the given name, and if none is there, then in system directories, such as **/bin** and **/usr/bin**. There is nothing special about system-provided commands except that they are kept in directories where the shell can find them. You can also keep commands in your own directories and instruct the shell to find them there. See the manual entry *sh*(1), under the sub-heading "Parameter Substitution," for the discussion of the **$PATH** shell environment variable.

The command name is the first word on an input line to the shell; the command and its arguments are separated from one another by space or tab characters.

When a program terminates, the shell will ordinarily regain control and give you back your prompt to indicate that it is ready for another command. The shell has many other capabilities, which are described in detail in *sh*(1).

The Current Directory

The UNIX System has a file system arranged in a hierarchy of directories. When you received your login ID, the system administrator also created a directory for you (ordinarily with the same name as your login ID, and known as your *login* or *home* directory). When you log in, that directory becomes your *current* or *working* directory, and any file name you type is, by default, assumed to be in that directory. Because you are the owner of this directory, you have full permissions to read, write, alter, or remove its contents.

Permissions to enter or modify other directories and files will have been granted or denied to you by their respective owners or by the system administrator. To change the current directory, use *cd*(1).

Path Names

To refer to files or directories not in the current directory, you must use a path name. Full path names begin with /, which is the name of the **root** directory of the whole file system. After the slash comes the name of each directory containing the next subdirectory (followed by a /), until finally the file or directory name is reached (e.g., **/usr/ae/filex** refers to file **filex** in directory **ae**, while **ae** is itself a subdirectory of **usr**, and **usr** is a subdirectory of the **root** directory). Use *pwd*(1) to print the full path name of the directory you are working in. See *intro*(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual* for a formal definition of *path name*.

If your current directory contains subdirectories, the path names of their respective files begin with the name of the corresponding subdirectory (*without* a prefixed /). A path name may be used anywhere a file name is required.

Important commands that affect files are *cp*(1), *mv* (see *cp*(1)), and *rm*(1), which respectively copy, move (i.e., rename), and remove files. To find out the status of files or directories, use *ls*(1). Use *mkdir*(1) for making directories and *rmdir* (see *rm*(1)) for removing them.

Text Entry and Display

Almost all text is entered through an editor. Common examples of UNIX System editors are *ed*(1) and *vi*(1). The commands most often used to print text on a terminal are *cat*(1), *pr*(1), and *pg*(1). The *cat*(1) command displays the contents of ASCII text files on the terminal, with no processing at all. The *pr*(1) command paginates the text, supplies headings, and has a facility for multi-column output. The *pg*(1) command displays text in successive portions no larger than your terminal screen.

Communicating With Others

Certain commands provide *inter-user* communication. Even if you do not plan to use them, it would be well to learn something about them because someone else may try to contact you. *mail*(1) or *mailx*(1) will leave a message whose presence will be announced to another user when he or she next logs in and at periodic intervals during the session. To communicate with another user currently logged in, *write*(1) is used. The corresponding entries in this manual also suggest how to respond to these two commands if you are their target.

TABLE OF CONTENTS

1. Commands

4. File Formats

5. Miscellaneous Facilities

7. Special Files

8. Additional System Administration Entries

NAME

intro – introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands (including system maintenance commands) available for your computer. The commands in this section should be used along with those listed in Sections 1, 2, 3, 4, and 5 of the *INTERACTIVE SDS Guide and Programmer's Reference Manual*. References of the form *name*(1), *name*(2), *name*(2P), *name*(3), *name*(3P), *name*(4), *name*(5), and *name*(5P) refer to entries in that manual. References of the form *name*(1), *name*(1M), *name*(1C), *name*(1G), *name*(1P), *name*(5), *name*(5P), *name*(7), and *name*(8) refer to entries in this manual. Entries of the form *name*(*n*P) refer to POSIX entries, which pertain to the operating system environment specified by U.S. Government standard *IEEE Std. 1003.1-1988*.

Manual Page Command Syntax

Unless otherwise noted, commands described in the **SYNOPSIS** section of a manual page accept options and other arguments according to the following syntax and should be interpreted as explained below.

name [–*option*...] [*cmdarg*...]
where:

[ ]	Surround an *option* or *cmdarg* that is not required.
...	Indicates multiple occurrences of the *option* or *cmdarg*.
name	The name of an executable file.
option	(Always preceded by a "–".) *noargletter*... or, *argletter optarg*[,...]
noargletter	A single letter representing an option without an option-argument. Note that more than one *noargletter* option can be grouped after one "–" (Rule 5 in the following text).
argletter	A single letter representing an option requiring an option-argument.
optarg	An option-argument (character string) satisfying a preceding *argletter*. Note that groups of *optargs* following an *argletter* must be separated by commas or separated by white space and quoted (Rule 8 below).
cmdarg	Path name (or other command argument) *not* beginning with "–", or "–" by itself indicating the standard input.

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new commands use them. *getopts*(1) should be used by all shell procedures to parse positional parameters and to check for legal options. It supports Rules 3-10 below. The enforcement of the other rules must be done by the command itself.

1. Command names (*name* above) must be between two and nine characters long.

2. Command names must include only lowercase letters and digits.
3. Option names (*option* above) must be one character long.
4. All options must be preceded by "–".
5. Options with no arguments may be grouped after a single "–".
6. The first option-argument (*optarg* above) following an option must be preceded by white space.
7. Option-arguments cannot be optional.
8. Groups of option-arguments following an option must either be separated by commas or separated by white space and quoted (e.g., –o xxx,z,yy or –o "xxx z yy").
9. All options must precede operands (*cmdarg* above) on the command line.
10. "––" may be used to indicate the end of the options.
11. The order of the options relative to one another should not matter.
12. The relative order of the operands (*cmdarg* above) may affect their significance in ways determined by the command with which they appear.
13. "–" preceded and followed by white space should only be used to mean standard input.

SEE ALSO

getopts(1).
exit(2), wait(2), getopt(3C) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

How to Get Started at the front of this document.

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 nonzero to indicate troubles such as erroneous parameters or bad or inaccessible data. It is called variously "exit code," "exit status," or "return code" and is described only where special conventions are involved.

BUGS

Regrettably, not all commands 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.

NAME

300, 300s – handle special functions of DASI 300 and 300s terminals

SYNOPSIS

300 [**+12**] [**−n**] [**−d**t,l,c]

300s [**+12**] [**−n**] [**−d**t,l,c]

DESCRIPTION

The *300* command supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal; *300s* performs the same functions for the DASI 300s (GSI 300s or DTC 300s) terminal. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. In the following discussion of the *300* command, it should be noted that unless your system contains the DOCUMENTER'S WORKBENCH Software, references to certain commands (e.g., *nroff*, *neqn*, *eqn,* etc.) will not work. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12-pitch text. It also reduces printing time 5 to 70%. The *300* command can be used to print equations neatly, in the sequence:

neqn file ... | nroff | 300

WARNING: if your terminal has a PLOT switch, make sure it is turned *on* before *300* is used.

The behavior of *300* can be modified by the optional flag arguments to handle 12-pitch text, fractional line spacings, messages, and delays.

+12 permits use of 12-pitch, 6 lines/inch text. DASI 300 terminals normally allow only two combinations: 10-pitch, 6 lines/inch, or 12-pitch, 8 lines/inch. To obtain the 12-pitch, 6 lines per inch combination, the user should turn the PITCH switch to 12, and use the **+12** option.

−n controls the size of half-line spacing. A half-line is, by default, equal to 4 vertical plot increments. Because each increment equals 1/48 of an inch, a 10-pitch line-feed requires 8 increments, while a 12-pitch line-feed needs only 6. The first digit of *n* overrides the default value, thus allowing for individual taste in the appearance of subscripts and superscripts. For example, *nroff* half-lines could be made to act as quarter-lines by using **−2**. The user could also obtain appropriate half-lines for 12-pitch, 8 lines/inch mode by using the option **−3** alone, having set the PITCH switch to 12-pitch.

−d*t*,*l*,*c* controls delay factors. The default setting is **−d3,90,30**. DASI 300 terminals sometimes produce peculiar output when faced with very long lines, too many tab characters, or long strings of blankless, non-identical characters. One null (delay) character is inserted in a line for every set of *t* tabs, and for every contiguous string of *c* non-blank, non-tab characters. If a line is longer than *l* bytes, 1+(total length)/20 nulls are inserted at the end of that line. Items can be omitted from the end of the list, implying use of the default values. Also, a value of zero for *t* (*c*) results in two null bytes per tab (character). The former may be needed for C programs, the latter for files like **/etc/passwd**. Because terminal behavior varies according to the specific characters printed and the load on a system, the user may have to experiment with these values to get correct output. The **−d** option exists only as a last resort for those few cases that do not otherwise print properly. For example, the file **/etc/passwd** may be printed using **−d3,30,5**. The value **−d0,1** is a good one to use for C programs that have many levels of indentation.

Note that the delay control interacts heavily with the prevailing carriage return and line-feed delays. The *stty*(1) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

The *300* command can be used with the *nroff* **−s** flag or **.rd** requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the following sequences are equivalent:

nroff −T300 files ... and nroff files ... | 300
nroff −T300−12 files ... and nroff files ... | 300 +12

The use of *300* can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of *300* may produce better aligned output.

SEE ALSO

450(1), mesg(1), graph(1G), stty(1), tabs(1), tplot(1G).

BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

NOTE

troff(1), *nroff*(1), and *eqn*(1) are not part of this UNIX system release.

NAME

4014 – paginator for the TEKTRONIX 4014 terminal

SYNOPSIS

4014 [−t] [−n] [−cN] [−pL] [file]

DESCRIPTION

The output of *4014* is intended for a TEKTRONIX 4014 terminal; *4014* arranges for 66 lines to fit on the screen, divides the screen into *N* columns, and contributes an eight-space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. TELETYPE Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page, *4014* waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command *!cmd* will send the *cmd* to the shell.

The command line options are:

−t Do not wait between pages (useful for directing output into a file).

−n Start printing at the current cursor position and never erase the screen.

−c*N* Divide the screen into *N* columns and wait after the last column.

−p*L* Set page length to *L*; *L* accepts the scale factors **i** (inches) and **l** (lines); default is lines.

SEE ALSO

pr(1).

NAME

450 — handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

The *450* command supports special functions of, and optimizes the use of, the DASI 450 terminal, or any terminal that is functionally identical, such as the Diablo 1620 or Xerox 1700. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols in the same manner as *300*(1). It should be noted that, unless your system contains DOCUMENTER'S WORKBENCH Software, certain commands (e.g., *eqn*, *nroff*, *tbl*, etc.) will not work. Use *450* to print equations neatly, in the sequence:

    neqn file ... | nroff | 450

WARNING: Make sure that the PLOT switch on your terminal is ON before *450* is used. The SPACING switch should be put in the desired position (either 10- or 12-pitch). In either case, vertical spacing is 6 lines/inch, unless dynamically changed to 8 lines per inch by an appropriate escape sequence.

Use *450* with the *nroff* **—s** flag or **.rd** requests when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the use of *450* can be eliminated in favor of one of the following:

    nroff —T450 files ...

or

    nroff —T450—12 files ...

The use of *450* can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of *450* may produce better aligned output.

SEE ALSO

300(1), mesg(1), stty(1), tabs(1), graph(1G), tplot(1G).

BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

NOTE

troff(1), *nroff*(1), and *eqn*(1) are not part of this UNIX system release.

NAME

accept, reject — allow or prevent LP requests

SYNOPSIS

/usr/lib/accept destinations
/usr/lib/reject [**−r**[reason]] destinations

DESCRIPTION

The *accept* command allows *lp*(1) to accept requests for the named *destinations*. A *destination* can be either a line printer (LP) or a class of printers. Use *lpstat*(1) to find the status of *destinations*.

The *reject* command 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

acct: acctdisk, acctdusg, accton, acctwtmp — overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk

/usr/lib/acct/acctdusg [**−u** file] [**−p** file]

/usr/lib/acct/accton [file]

/usr/lib/acct/acctwtmp "reason"

DESCRIPTION

Accounting software is structured as a set of tools (consisting of both C programs and shell procedures) that can be used to build accounting systems. When the system is installed, accounting is initially in the "off" state. *acctsh*(1M) describes the set of shell procedures built on top of the C programs.

Connect time accounting is handled by various programs that write records into **/etc/utmp**, as described in *utmp*(4). The programs described in *acctcon*(1M) convert this file into session and charging records, which are then summarized by *acctmerg*(1M).

Process accounting is performed by the UNIX system kernel. Upon termination of a process, one record per process is written to a file (normally **/usr/adm/pacct**). The programs in *acctprc*(1M) summarize this data for charging purposes; *acctcms*(1M) is used to summarize command usage. Current process data may be examined using *acctcom*(1).

Process accounting and connect time accounting [or any accounting records in the format described in *acct*(4)] can be merged and summarized into total accounting records by *acctmerg* [see **tacct** format in *acct*(4)]. *prtacct* [see *acctsh*(1M)] is used to format any or all accounting records.

acctdisk reads lines that contain user ID, login name, and number of disk blocks and converts them to total accounting records that can be merged with other accounting records.

acctdusg reads its standard input (usually from **find / −print**) and computes disk resource consumption (including indirect blocks) by login. If **−u** is given, records consisting of those file names for which *acctdusg* charges no one are placed in *file* (a potential source for finding users trying to avoid disk charges). If **−p** is given, *file* is the name of the password file. This option is not needed if the password file is **/etc/passwd**. [See *diskusg*(1M) for more details.]

accton alone turns process accounting off. If *file* is given, it must be the name of an existing file to which the kernel appends process accounting records [see *acct*(2) and *acct*(4)].

acctwtmp writes a *utmp*(4) record to its standard output. The record contains the current time and a string of characters that describe the *reason*. A record type of ACCOUNTING is assigned [see *utmp*(4)]. *Reason* must be a string of 11 or fewer characters, numbers, **$**, or spaces. For example, the following are suggestions for use in reboot and shutdown procedures, respectively:

```
acctwtmp uname >> /etc/wtmp
acctwtmp "file save" >> /etc/wtmp
```

FILES

/etc/passwd	used for login name to user ID conversions
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/pacct	current process accounting file
/etc/wtmp	login/logoff history file

SEE ALSO

acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), diskusg(1M), fwtmp(1M), runacct(1M).

acct(2), acct(4), utmp(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

acctcms – command summary from per-process accounting records

SYNOPSIS

/usr/lib/acct/acctcms [options] files

DESCRIPTION

acctcms reads one or more *files*, normally in the form described in *acct*(4). It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format. The *options* are:

–a Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, "hog factor", characters transferred, and blocks read and written, as in *acctcom*(1). Output is normally sorted by total kcore-minutes.

–c Sort by total CPU time, rather than total kcore-minutes.

–j Combine all commands invoked only once under "***other".

–n Sort by number of command invocations.

–s Any file names encountered hereafter are already in internal summary format.

–t Process all records as total accounting records. The default internal summary format splits each field into prime and non-prime time parts. This option combines the prime and non-prime time parts into a single field that is the total of both, and provides upward compatibility with old (i.e., UNIX System V) style *acctcms* internal summary format records.

The following options may be used only with the **-a** option.

–p Output a prime-time-only command summary.

–o Output a non-prime (offshift) time only command summary.

When **–p** and **–o** are used together, a combination prime and non-prime time report is produced. All the output summaries will be total usage except number of times executed, CPU minutes, and real minutes which will be split into prime and non-prime.

A typical sequence for performing daily command accounting and for maintaining a running total is:

```
acctcms file ... >today
cp total previoustotal
acctcms –s today previoustotal >total
acctcms –a –s today
```

SEE ALSO

acct(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M).

acct(2), acct(4), utmp(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

BUGS

Unpredictable output results if **−t** is used on new style internal summary format files, or if it is not used with old style internal summary format files.

NAME

acctcom − search and print process accounting file(s)

SYNOPSIS

acctcom [[options] [file]] . . .

DESCRIPTION

acctcom reads *file*, the standard input, or **/usr/adm/pacct**, in the form described by *acct*(4) and writes selected records to the standard output. Each record represents the execution of one process. The output shows the **COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SEC), CPU (SEC), MEAN SIZE(K)**, and optionally, **F** (the *fork/exec* flag: **1** for *fork* without *exec*), **STAT** (the system exit status), **HOG FACTOR, KCORE MIN, CPU FACTOR, CHARS TRNSFD,** and **BLOCKS READ** (total blocks read and written).

The command name is prepended with a **#** if it was executed with *super-user* privileges. If a process is not associated with a known terminal, a **?** is printed in the **TTYNAME** field.

If no *files* are specified, and if the standard input is associated with a terminal or **/dev/null** (as is the case when using **&** in the shell), **/usr/adm/pacct** is read; otherwise, the standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward, i.e., in chronological order by process completion time. The file **/usr/adm/pacct** is usually the current file to be examined; a busy system may need several such files of which all but the current file are found in **/usr/adm/pacct***?*. The *options* are:

−a Show some average statistics about the processes selected. The statistics will be printed after the output records.

−b Read backwards, showing latest commands first. This *option* has no effect when the standard input is read.

−f Print the *fork/exec* flag and system exit status columns in the output.

−h Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:
(total CPU time)/(elapsed time).

−i Print columns containing the I/O counts in the output.

−k Instead of memory size, show total kcore-minutes.

−m Show mean core size (the default).

−r Show CPU factor (user time/(system-time + user-time).

−t Show separate system and user CPU times.

−v Exclude column headings from the output.

−l *line* Show only processes belonging to terminal **/dev/***line*.

−u *user* Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a **#** which designates only those processes executed with *super-user* privileges, or **?** which designates only those processes associated with unknown user IDs.

−g *group* Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.

−s *time* Select processes existing at or after *time*, given in the format *hr* [:*min* [:*sec*]].

−e *time* Select processes existing at or before *time*.

−S *time* Select processes starting at or after *time*.

−E *time* Select processes ending at or before *time*. Using the same *time* for both **−S** and **−E** shows the processes that existed at *time*.

−n *pattern* Show only commands matching *pattern* that may be a regular expression as in *ed*(1) except that **+** means one or more occurrences.

−q Do not print any output records; just print the average statistics as with the **−a** option.

−o *ofile* Copy selected process records in the input data format to *ofile*; suppress standard output printing.

−H *factor* Show only processes that exceed *factor*, where factor is the "hog factor" as explained in option **−h** above.

−O *sec* Show only processes with CPU system time exceeding *sec* seconds.

−C *sec* Show only processes with total CPU time, system plus user, exceeding *sec* seconds.

−I *chars* Show only processes transferring more characters than the cut-off number given by *chars*.

FILES

/etc/passwd
/usr/adm/pacct
/etc/group

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), ps(1), runacct(1M), su(1M).

acct(2), acct(4), utmp(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

BUGS

acctcom reports only on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time, then *time* is interpreted as occurring on the previous day.

NAME

acctcon: acctcon1, acctcon2 – connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]

/usr/lib/acct/acctcon2

DESCRIPTION

acctcon1 converts a sequence of login/logoff records read from its standard input to a sequence of records, one per login session. Its input should normally be redirected from **/etc/wtmp**. Its output is ASCII, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time. The *options* are:

–p Print input only, showing line name, login name, and time (in both numeric and date/time formats).

–t *acctcon1* maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The **–t** flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.

–l *file* *File* is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Hang-up, termination of *login*(1) and termination of the login shell each generate logoff records, so that the number of logoffs is often three to four times the number of sessions. See *init*(1M) and *utmp*(4).

–o *file* *File* is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

acctcon2 expects as input a sequence of login session records and converts them into total accounting records [see **tacct** format in *acct*(4)].

EXAMPLES

These commands are typically used as shown below. The file **ctmp** is created only for the use of *acctprc*(1M) commands:

```
acctcon1 –t –l lineuse –o reboots <wtmp | sort +1n +2 >ctmp
acctcon2 <ctmp | acctmerg >ctacct
```

FILES

/etc/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), init(1M), runacct(1M).
acct(2), acct(4), utmp(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

BUGS

The line usage report is confused by date changes. Use *wtmpfix* [see *fwtmp*(1M)] to correct this situation.

NAME

acctmerg – merge or add total accounting files

SYNOPSIS

/usr/lib/acct/acctmerg [options] [file] . . .

DESCRIPTION

acctmerg reads its standard input and up to nine additional files, all in the **tacct** format [see *acct*(4)] or an ASCII version thereof. It merges these inputs by adding records whose keys (normally user ID and name) are identical, and expects the inputs to be sorted on those keys. *Options* are:

- **–a** Produce output in ASCII version of **tacct**.
- **–i** Input files are in ASCII version of **tacct**.
- **–p** Print input with no processing.
- **–t** Produce a single record that totals all input.
- **–u** Summarize by user ID, rather than user ID and name.
- **–v** Produce output in verbose ASCII format, with more precise notation for floating point numbers.

EXAMPLES

The following sequence is useful for making "repairs" to any file kept in this format:

```
acctmerg –v <file1 >file2
        edit file2 as desired ...
acctmerg –i <file2 >file1
```

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M).

acct(2), acct(4), utmp(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

acctprc: acctprc1, acctprc2 – process accounting

SYNOPSIS

/usr/lib/acct/acctprc1 [ctmp]

/usr/lib/acct/acctprc2

DESCRIPTION

acctprc1 reads input in the form described by *acct*(4), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in memory segment units). If **ctmp** is given, it is expected to contain a list of login sessions, in the form described in *acctcon*(1M), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in **ctmp** helps it distinguish among different login names that share the same user ID.

acctprc2 reads records in the form written by *acctprc1*, summarizes them by user ID and name, then writes the sorted summaries to the standard output as total accounting records.

These commands are typically used as shown below:

```
acctprc1 ctmp </usr/adm/pacct | acctprc2 >ptacct
```

FILES

/etc/passwd

SEE ALSO

acct(1M), acctcms(1M), acctcom(1) acctcon(1M), acctmerg(1M), acctsh(1M), cron(1M), fwtmp(1M), runacct(1M).

acct(2), acct(4), utmp(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

BUGS

Although it is possible to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from *cron*(1M), for example. More precise conversion can be done by faking login sessions on the console via the *acctwtmp* program in *acct*(1M).

CAVEAT

A memory segment of the mean memory size is a unit of measure for the number of bytes in a logical memory segment on a particular processor. For example, on a PDP-11/70 this measure would be in 64-byte units, while on a VAX11/780 it would be in 512-byte units.

NAME

acctsh: chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct – shell procedures for accounting

SYNOPSIS

/usr/lib/acct/chargefee login-name number

/usr/lib/acct/ckpacct [blocks]

/usr/lib/acct/dodisk [-o] [files ...]

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct number

/usr/lib/acct/nulladm file

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily [-l] [-c] [mmdd]

/usr/lib/acct/prtacct file ["heading"]

/usr/lib/acct/runacct [mmdd] [mmdd state]

/usr/lib/acct/shutacct ["reason"]

/usr/lib/acct/startup

/usr/lib/acct/turnacct on | off | switch

DESCRIPTION

chargefee can be invoked to charge a *number* of units to *login-name*. A record is written to **/usr/adm/fee** to be merged with other accounting records during the night.

ckpacct should be initiated via *cron*(1M). It periodically checks the size of **/usr/adm/pacct**. If the size exceeds *blocks*, 1000 by default, *turnacct* will be invoked with argument **switch**. If the number of free disk blocks in the **/usr** file system falls below 500, *ckpacct* will automatically turn off the collection of process accounting records via the *off* argument to *turnacct*. When at least this number of blocks is restored, the accounting will be activated again. This feature is sensitive to the frequency at which *ckpacct* is executed, usually by *cron*.

dodisk should be invoked by *cron* to perform the disk accounting functions. By default, it will do disk accounting on the special files in **/etc/fstab.** If the **—o** flag is used, it will do a slower version of disk accounting by login directory. *Files* specify the one or more file system names where disk accounting will be done. If *files* are used, disk accounting will be done on these file systems only. If the **—o** flag is used, *files* should be mount points of mounted file system. If omitted, they should be the special file names of mountable file systems.

lastlogin is invoked by *runacct* to update **/usr/adm/acct/sum/loginlog,** which shows the last date on which each person logged in.

monacct should be invoked once each month or each accounting period. *Number* indicates which month or period it is. If *number* is not given, it defaults to the current month (01–12). This default is useful if *monacct* is to executed via *cron*(1M) on the first day of each month. *monacct* creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum**.

nulladm creates *file* with mode 664 and ensures that owner and group are **adm**. It is called by various accounting shell procedures.

prctmp can be used to print the session record file (normally **/usr/adm/acct/nite/ctmp** created by *acctcon*(1M).

prdaily is invoked by *runacct* to format a report of the previous day's accounting data. The report resides in **/usr/adm/acct/sum/rprt***mmdd* where *mmdd* is the month and day of the report. The current daily accounting reports may be printed by typing **prdaily**. Previous days' accounting reports can be printed by using the *mmdd* option and specifying the exact report date desired. The **−l** flag prints a report of exceptional usage by login id for the specified date. Previous daily reports are cleaned up and therefore inaccessible after each invocation of *monacct*. The **−c** flag prints a report of exceptional resource usage by command, and may be used on current day's accounting data only.

prtacct can be used to format and print any total accounting (**tacct**) file.

runacct performs the accumulation of connect, process, fee, and disk accounting on a daily basis. It also creates summaries of command usage. For more information, see *runacct*(1M).

shutacct is invoked during a system shutdown to turn process accounting off and append a "reason" record to **/etc/wtmp**.

startup is called by **/etc/init.d/acct** to turn the accounting on whenever the system is brought to a multiuser state.

turnacct is an interface to *accton* [see *acct*(1M)] to turn process accounting **on** or **off**. The **switch** argument turns accounting off, moves the current **/usr/adm/pacct** to the next free name in **/usr/adm/pacct***incr* (where *incr* is a number starting with **1** and incrementing by one for each additional **pacct** file), then turns accounting back on again. This procedure is called by *ckpacct* and thus can be taken care of by the *cron* and used to keep **pacct** to a reasonable size. *acct* starts and stops process accounting via *init* and *shutdown* accordingly.

FILES

/usr/adm/fee	accumulator for fees
/usr/adm/pacct	current file for per-process accounting
/usr/adm/pacct*	used if pacct gets large and during execution of daily accounting procedure
/etc/wtmp	login/logoff summary
/usr/lib/acct/ptelus.awk	contains the limits for exceptional usage by login id
/usr/lib/acct/ptecms.awk	contains the limits for exceptional usage by command name
/usr/adm/acct/nite	working directory
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/acct/sum	summary directory, should be saved

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), cron(1M), diskusg(1M), fwtmp(1M), runacct(1M).

acct(2), acct(4), utmp(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

adduser – create a login for a new user

SYNOPSIS

adduser loginid name userid logdir Yes/No

DESCRIPTION

Adduser is used to create a login for a new user and must be given the following arguments:

loginid This is the user's login name. It can be no longer than eight alphanumeric characters.

name This is the user's full name. It identifies the person to whom the login is assigned. If *name* contains spaces, it must be in quotes.

userid This is the numerical user id that will be associated with the *loginid*. It must be between 100 and 50000 and must be unique for each user.

logdir This is the user's home directory. It must be a valid directory name and cannot already exist on the system. Typically the user's home directory matches the login name. For example, the login "ams" might have a home directory of **/usr/ams**.

Yes/No If the user will have system administration privileges.

NAME

at, batch – execute commands at a later time

SYNOPSIS

at time [date] [+ increment]
at –r job ...
at -l [job ...]

batch

DESCRIPTION

The *at* and *batch* commands 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. *at* may be used with the following options:

–r Removes jobs previously scheduled with *at*.

–l 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 name appears 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 **at.deny** is empty, global usage is permitted. The allow/deny files consist of one user name per line. These files can only be modified by the super-user.

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.

The **at −r** command 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 remove only your own jobs unless you are the super-user.

EXAMPLES

The *at* and *batch* commands read from standard input the commands to be executed at a later time. *sh*(1) provides a different way of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

batch
sort *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 <<!
sort *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

/usr/lib/cron	main cron directory
/usr/lib/cron/at.allow	list of allowed users
/usr/lib/cron/at.deny	list of denied users
/usr/lib/cron/queue	scheduling information
/usr/spool/cron/atjobs	spool area

SEE ALSO

cron(1M), kill(1), mail(1), nice(1), ps(1), sh(1), sort(1).

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

awk − pattern scanning and processing language

SYNOPSIS

awk [**−Fc**] [prog] [parameters] [files]

DESCRIPTION

The *awk* language 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) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*].

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument is present. 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.

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 *relop* is any of the six relational operators in C, and *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), sed(1).
lex(1), printf(3S) in the *INTERACTIVE SDS Guide and Programmer's Reference 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.

NAME

backup – performs backup functions

SYNOPSIS

backup [**–t**] [**–p** | **–c** | **–f** <files> | **–u** "<user1> [user2]"] **–d** <device>

backup –h

DESCRIPTION

–h produces a history of backups. Tells the user when the last complete and incremental/partial backups were done.

–c complete backup. All files changed since the system was installed are backed up.

–p incremental/partial backup. If an incremental/partial backup was done, all files modified since that time are backed up, otherwise all files modified since the last complete backup are backed up. A complete backup must be done before a partial backup.

–f backup files specified by the *<files>* argument. File names may contain characters to be expanded (i.e., *, .) by the shell. The argument must be in quotes.

–u backup a user's home directory. All files in the user's home directory will be backed up. At least one user must be specified but it can be more. The argument must be in quotes if more than one user is specified. If the user name is "all", then all the user's home directories will be backed up.

–d used to specify the device to be used. It defaults to **/dev/rdsk/f0q15d** (the 1.2M floppy).

–t used when the device is a tape. This option must be used with the **–d** option when the tape device is specified.

A complete backup must be done before a partial backup can be done. Raw devices rather than block devices should always be used. The program can handle multi-volume backups. The program will prompt the user when it is ready for the next medium. The program will give you an estimated number of floppies/tapes that will be needed to do the backup. Floppies MUST be formatted before the backup is done. Tapes do not need to be formatted. If backup is done to tape, the tape must be rewound.

SEE ALSO

qt(7).

NAME

banner – make posters

SYNOPSIS

banner strings

DESCRIPTION

The *banner* command prints its arguments (each up to 10 characters long) in large letters on the standard output. Spaces can be included in an argument by surrounding it with quotes. The maximum number of characters that can be accommodated in a line is implementation-dependent; excess characters are simply ignored.

SEE ALSO

echo(1).

NAME

basename, dirname — deliver portions of path names

SYNOPSIS

basename string [suffix]
dirname string

DESCRIPTION

The *basename* command 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.

The *dirname* command 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).

NAME

bc — arbitrary-precision arithmetic language

SYNOPSIS

bc [−c] [−l] [file ...]

DESCRIPTION

The *bc* command 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 *bc*(1) utility is actually a preprocessor for *dc*(1), which it invokes automatically unless the **−c** option is present. In this case the *dc* input is sent to the standard output instead. The options are as follows:

−c Compile only. The output is send to the standard output.

−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.

EXAMPLE

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==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

The *bc* command does not yet recognize the logical operators, **&&** and **||**.
For statement must have all three expressions (E's).
Quit is interpreted when read, not when executed.

NAME

bdiff − big diff

SYNOPSIS

bdiff file1 file2 [n] [**−s**]

DESCRIPTION

The *bdiff* command is used in a manner analogous to *diff*(1) to find which lines in two files must be changed to bring the files into agreement. Its purpose is to allow processing of files which are too large for *diff*.

The parameters to *bdiff* are:

file1 (file2)
: The name of a file to be used. If *file1 (file2)* is −, the standard input is read.

n
: The number of line 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.

−s
: Specifies that no diagnostics are to be printed by *bdiff* (silent option). Note, however, that this does not suppress possible diagnostic messages from *diff*(1), which *bdiff* calls.

The *bdiff* command 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. 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).

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 and 32K lines, with up to 512 characters per line, including new-line (255 for 16-bit machines). *bfs* is usually more efficient than *ed*(1) 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*(1). 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*(1) 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*(1). 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 (readable and writable by everyone), unless your *umask* setting [see *umask*(1)] dictates otherwise. 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**:

```
1,%5p
1,%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 \.

```
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
: l
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn l
xv45
: l
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz l
```

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), umask(1).

DIAGNOSTICS

? for errors in commands if prompting is turned off. Self-explanatory error messages when prompting is on.

NAME

boot – UNIX System boot program

DESCRIPTION

The *boot* program is responsible for detecting the type of machine and its complement of hardware, setting up the environment necessary for the execution of the UNIX System kernel, and loading and executing the kernel. The *boot* program does not perform these functions alone; the booting process demands the cooperation of a number of pieces of software. The booting process varies slightly depending on whether the kernel resides on a diskette or fixed disk.

The booting process begins with a machine in the reset state. This may occur because the machine has just been turned on or because a reboot sequence has been entered. From this point, the ROM BIOS supplied with the machine gains control and goes through its power-on self test (POST) sequence. After this, it will attempt to boot an operating system off of a diskette if one is present. Failing that, it will attempt to boot from the fixed disk.

When booting from a diskette, the ROM BIOS reads sector 1 from track 0 of cylinder 0 (the first logical sector on the disk) and jumps to the beginning of that block. *boot* must then read in the rest of its program and continue the booting process. The *boot* program may occupy the entire first cylinder on a 5.25-inch or 3.5-inch high density diskette, or the first two cylinders on a 3.5-inch double density (700Kb) diskette. This limits the diskette-based *boot* program to a size of 18Kb.

Booting from a fixed disk is a more complex operation. The ROM BIOS reads in the first logical sector of the disk. This sector contains a small fragment of code called the *masterboot* and a table of partitions called the *fdisk* table. The *fdisk* table has up to four valid entries, and each entry describes a portion of the disk by giving its beginning cylinder, head, and sector number, ending cylinder, head, and sector, and the logical starting and ending sector numbers. At most, one partition is marked as the *active* partition.

Note that the logical sector numbers are maintained in 32-bit words, which is sufficient for an extremely large disk (over a terabyte); however, the other values are packed into small fields which limit their ranges: cylinder values are limited to 1024, heads to 16, and sectors to 64. This means that on some disks, it may be possible to set up a UNIX System partition that cannot be booted from.

The *masterboot* program searches the *fdisk* table for the active partition, and reads in the first logical sector of that partition. This will be the first block of the *boot* program when the active partition contains the INTERACTIVE UNIX Operating System. As is the case with the diskette version, this first block must then read in the remainder of the *boot* program. Because the UNIX System VTOC structure is located in sector 29 of the partition, the fixed disk *boot* program cannot be any larger than 14.5Kb.

After the *boot* program is read in from either a diskette or a fixed disk, *boot* attempts to determine which machine it is on. It does this by reading another program, **/etc/initprog/identify**, and executing it.

This program performs a number of tests, often searching the ROM BIOS for identifying strings, until a match is found for a particular machine or until all tests fail. In the latter case, the machine is assumed to be a generic PC AT.

As a part of determining the machine type, a particular machine initialization program (or *initprog*) is selected. These programs reside in **/etc/initprog**. The selected program is then loaded and executed. It is responsible for doing any machine-dependent operations that may be necessary. Certain generic operations such as sizing memory or detecting the presence of floating point coprocessors can also be done at this time if they require some nonstandard code.

After the execution of the machine-dependent *initprog*, a machine-independent initialization program, **/etc/initprog/machsetup**, is run. If not done by the dependent *initprog*, *machsetup* determines the location and size of memory, the number and types of fixed disks and diskettes, and a few other pieces of information.

All of the information determined by these programs is saved in the *bootinfo* structure, where it is used by the kernel and drivers. It is also available to user programs via a *sysi86* system call if needed.

At this time, the UNIX System kernel, **/unix**, is loaded. As the *boot* program starts to load **/unix**, it displays the message:

Booting the UNIX system...
Loading file /unix
Strike any key to interrupt

During this time the user can interrupt the *boot* in order to specify a different kernel by typing any key. The *boot* will respond with:

Enter the name of a kernel to boot:

The user should enter a UNIX System path name. The file specified must reside within the **root** file system; the *boot* program is unable to search other file systems. If the file is not found, *boot* prints a message and asks for another file name. If no response is made within 30 seconds or if the user responds with a carriage return, *boot* will once again attempt to load **/unix** and execute it.

Once a kernel is determined, it is loaded into memory. The *boot* program then generates the appropriate page directory and page tables for the kernel, sets up an interrupt descriptor table and a global descriptor table, turns on paging, and does a long jump through a task switch selector into the kernel.

The *boot* program is installed on a fixed disk by the *mkpart*(1) program when it initializes the disk. If *boot* becomes damaged, a fresh copy is maintained in **/etc/boot**, and *mkpart* can be used (with the **−b** option) to update it. Part of the installation of a *boot* into the UNIX System partition includes stamping the number of sectors per track into it; this makes the installed *boot* unusable on other disks. However, **/etc/boot** itself is not modified.

FILES

/etc/boot
/etc/initprog/identify
/etc/initprog/machsetup
/etc/initprog/*

SEE ALSO

fdisk(1M), init(1M), mkpart(1M), fd(7).

DIAGNOSTICS

The *masterboot* and *boot* programs display different error messages. The *masterboot* program displays an error message and locks the system. The following is a list of the most common *masterboot* messages and their meanings:

IO ERR An error occurred when trying to read in the partition boot of the active operating system.

BAD TBL The bootable partition indicator of at least one of the operating systems in the *fdisk* table contains an unrecognizable code.

NO OS There was an unrecoverable error after trying to execute the active operating system's partition boot.

If *boot* displays a message other than the ones shown above, it is a fatal error and must be corrected before the kernel can be booted. The messages are intended to be self-explanatory, but are they technically detailed. If you encounter one, be sure to copy it completely and accurately and contact your vendor for technical support.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

brc, bcheckrc — system initialization procedures

SYNOPSIS

/etc/brc

/etc/bcheckrc

DESCRIPTION

These shell procedures are executed via entries in **/etc/inittab** by *init*(1M) whenever the system is booted (or rebooted).

First, the *bcheckrc* procedure checks the status of the root file system. If the root file system is found to be bad, *bcheckrc* repairs it.

Then, the *brc* procedure clears the mounted file system table, **/etc/mnttab,** and puts the entry for the root file system into the mount table.

After these two procedures have executed, *init* checks for the *initdefault* value in **/etc/inittab**. This tells *init* in which run level to place the system. Since *initdefault* is initially set to **2**, the system will be placed in the multi-user state via the **/etc/rc2** procedure.

Note that *bcheckrc* should always be executed before *brc*. Also, these shell procedures may be used for several run-level states.

SEE ALSO

fsck(1M), init(1M), rc2(1M), shutdown(1M).

NAME

cal – print calendar

SYNOPSIS

cal [[month] year]

DESCRIPTION

The *cal* command 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 the United States.

EXAMPLES

An unusual calendar is printed for September 1752. That is the month 11 days were skipped to make up for lack of leap year adjustments. To see this calendar, type: **cal 9 1752**

BUGS

The year is always considered to start in January even though this is historically naive.

Beware that "cal 88" refers to the early Christian era, not the 20th century.

NAME

calendar – reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

The *calendar* command 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 his or her 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

captoinfo – convert a termcap description into a terminfo description

SYNOPSIS

captoinfo [**–v** ...] [**–V**] [**–1**] [**–w** width] file ...

DESCRIPTION

The *captoinfo* command looks in *file* for *termcap* descriptions. For each one found, an equivalent *terminfo*(4) description is written to standard output, along with any comments found. A description which is expressed as relative to another description (as specified in the *termcap tc=* field) will be reduced to the minimum superset before being output.

If no *file* is given, then the environment variable **TERMCAP** is used for the file name or entry. If **TERMCAP** is a full path name to a file, only the terminal whose name is specified in the environment variable **TERM** is extracted from that file. If the environment variable **TERMCAP** is not set, then the file **/etc/termcap** is read.

–v print out tracing information on standard error as the program runs. Specifying additional **–v** options will cause more detailed information to be printed.

–V print out the version of the program in use on standard error and exit.

–1 cause the fields to print out, one to a line. Otherwise, the fields will be printed several to a line, up to a maximum width of 60 characters.

–w change the output to *width* characters.

FILES

/usr/lib/terminfo/?/* compiled terminal description data base

CAVEATS

Certain *termcap* defaults are assumed to be true. For example, the bell character (*terminfo bel*) is assumed to be ^*G*. The linefeed capability (*termcap nl*) is assumed to be the same for both *cursor_down* and *scroll_forward* (*terminfo cud1* and *ind*, respectively.) Padding information is assumed to belong at the end of the string.

The algorithm used to expand parameterized information for *termcap* fields such as *cursor_position* (*termcap cm*, *terminfo cup*) will sometimes produce a string which, though technically correct, may not be optimal. In particular, the rarely used *termcap* operation %**n** will produce strings that are especially long. Most occurrences of these non-optimal strings will be flagged with a warning message and may need to be recoded by hand.

The short two-letter name at the beginning of the list of names in a *termcap* entry, a hold-over from an earlier version of the UNIX system, has been removed.

DIAGNOSTICS

tgetent failed with return code n (reason).

The termcap entry is not valid. In particular, check for an invalid 'tc=' entry.

unknown type given for the termcap code *cc*.
: The termcap description had an entry for *cc* whose type was not Boolean, numeric, or string.

wrong type given for the Boolean (numeric, string) termcap code *cc*.
: The Boolean *termcap* entry *cc* was entered as a numeric or string capability.

the Boolean (numeric, string) termcap code *cc* is not a valid name.
: An unknown *termcap* code was specified.

tgetent failed on TERM=term.
: The terminal type specified could not be found in the *termcap* file.

TERM=term: **cap** *cc* (**info** *ii*) is NULL: REMOVED
: The *termcap* code was specified as a null string. The correct way to cancel an entry is with an '@', as in ':bs@:'. Giving a null string could cause incorrect assumptions to be made by the software which uses *termcap* or *terminfo*.

a function key for *cc* was specified, but it already has the value *vv*.
: When parsing the **ko** capability, the key *cc* was specified as having the same value as the capability *cc*, but the key *cc* already had a value assigned to it.

the unknown termcap name *cc* was specified in the **ko** termcap capability.
: A key was specified in the **ko** capability which could not be handled.

the *vi* character *v* (**info** *ii*) has the value *xx*, but **ma** gives *n*.
: The **ma** capability specified a function key with a value different from that specified in another setting of the same key.

the unknown *vi* key *v* was specified in the **ma** termcap capability.
: A *vi*(1) key unknown to *captoinfo* was specified in the **ma** capability.

Warning: *termcap* **sg** (*nn*) and *termcap* **ug** (*nn*) had different values.
: *terminfo* assumes that the **sg** (now **xmc**) and **ug** values were the same.

Warning: the string produced for *ii* may be inefficient.
: The parameterized string being created should be rewritten by hand.

Null termname given.
: The terminal type was null. This is given if the environment variable **TERM** is not set or is null.

cannot open *file* for reading.
: The specified file could not be opened.

SEE ALSO

infocmp(1M), tic(1M).
curses(3X), terminfo(4) in the *INTERACTVE SDS Guide and Programmer's Reference Manual*.
Chapter 10 in the *Programmer's Guide*.

NOTES

The *captoinfo* command should be used to convert *termcap* entries to *terminfo*(4) entries because the *termcap* data base (from earlier versions of UNIX System V) may not be supplied in future releases.

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 **file** on your terminal, and:

cat file1 file2 >file3

concatenates **file1** and **file2**, and writes the results in **file3**.

If no input file is given, or if the argument − is encountered, *cat* reads from the standard input file.

The following options apply to *cat*:

−u The output is not buffered. (The default is buffered output.)

−s *cat* is silent about nonexistent files.

−v Causes non-printing characters (with the exception of tabs, new-lines and form-feeds) to be printed visibly. ASCII control characters (octal 000 − 037) are printed as ^*n*, where *n* is the corresponding ASCII character in the range octal 100 − 137 (@, A, B, C, . . ., X, Y, Z, [, \,], ^, and _); the DEL character (octal 0177) is printed ^?. Other non-printable characters are printed as **M-***x*, where *x* is the ASCII character specified by the low-order seven bits.

The following options may be used with the **−v** option:

−t Causes tabs to be printed as ^*I*'s and formfeeds to be printed as ^*L*'s.

−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

Redirecting the output of *cat* onto one of the files being read will cause the loss of the data originally in the file being read. For example, typing:

cat file1 file2 >file1

will cause the original data in **file1** to be lost.

SEE ALSO

cp(1), pg(1), pr(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 *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

chmod – change mode

SYNOPSIS

chmod mode file ...

chmod mode directory ...

DESCRIPTION

The permissions of the named *files* or *directories* are changed according to mode, which may be symbolic or absolute. Absolute changes to permissions are stated using octal numbers:

chmod *nnn file(s)*

where *n* is a number from 0 to 7. Symbolic changes are stated using mnemonic characters:

chmod *a operator b file(s)*

where *a* is one or more characters corresponding to **user**, **group**, or **other**; where *operator* is +, –, and =, signifying assignment of permissions; and where *b* is one or more characters corresponding to type of permission.

An absolute mode is given as an octal number constructed from the OR of the following modes:

4000	set user ID on execution
20#0	set group ID on execution if # is **7**, **5**, **3**, or **1**
	enable mandatory locking if # is **6**, **4**, **2**, or **0**
1000	sticky bit is turned on [see *chmod*(2)]
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

Symbolic changes are stated using letters that correspond both to access classes and to the individual permissions themselves. Permissions to a file may vary depending on your user identification number (UID) or group identification number (GID). Permissions are described in three sequences each having three characters:

User	Group	Other
rwx	rwx	rwx

This example (meaning that **u**ser, **g**roup, and **o**thers all have **r**eading, **w**riting, and e**x**ecution permission to a given file) demonstrates two categories for granting permissions: the access class and the permissions themselves.

Thus, to change the mode of a file's (or directory's) permissions using *chmod*'s symbolic method, use the following syntax for mode:

[who] operator [permission(s)], ...

A command line using the symbolic method would appear as follows:

chmod g+rw *file*

This command would make *file* readable and writable by the group.

The *who* part can be stated as one or more of the following letters:

u	user's permissions
g	group's permissions
o	others permissions

The letter **a** (all) is equivalent to **ugo** and is the default if *who* is omitted.

Operator can be + to add *permission* to the file's mode, − to take away *permission*, or = to assign *permission* absolutely. (Unlike other symbolic operations, = has an absolute effect in that it resets all other bits.) Omitting *permission* is only useful with = to take away all permissions.

Permission is any compatible combination of the following letters:

r	reading permission
w	writing permission
x	execution permission
s	user or group set-ID is turned on
t	sticky bit is turned on
l	mandatory locking will occur during access

Multiple symbolic modes separated by commas may be given, though no spaces may intervene between these modes. Operations are performed in the order given. Multiple symbolic letters following a single operator cause the corresponding operations to be performed simultaneously. The letter **s** is only meaningful with **u** or **g**, and **t** only works with **u**.

Mandatory file and record locking (**l**) refers to a file's ability to have its reading or writing permissions locked while a program is accessing that file. It is not possible to permit group execution and enable a file to be locked on execution at the same time. In addition, it is not possible to turn on the set-group-ID and enable a file to be locked on execution at the same time. The following examples,

chmod g+x,+l *file*
chmod g+s,+l *file*

are, therefore, illegal usages and will elicit error messages.

Only the owner of a file or directory (or the super-user) may change a file's mode. Only the super-user may set the sticky bit on a non-directory file. In order to turn on a file's set-group-ID, your own group ID must correspond to the file's, and group execution must be set.

EXAMPLES

chmod a−x *file*

chmod 444 *file*

The first examples deny execution permission to all. The absolute (octal) example permits only reading permissions.

chmod go+rw *file*

chmod 606 *file*

These examples make a file readable and writable by the group and others.

chmod +l *file*

This causes a file to be locked during access.

chmod =rwx,g+s *file*

chmod 2777 *file*

These last two examples enable all to read, write, and execute the file; and they turn on the set-group-ID.

SEE ALSO

ls(1).
chmod(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

chown, chgrp – change owner or group

SYNOPSIS

chown owner file ...

chown owner directory ...

chgrp group file ...

chgrp group directory ...

DESCRIPTION

The *chown* command changes the owner of the *files* or *directories* to *owner*. The owner may be either a decimal user ID or a login name found in the password file.

The *chgrp* command changes the group ID of the *files* or *directories* 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.

Only the owner of a file (or the super-user) may change the owner or group of that file.

FILES

/etc/passwd
/etc/group

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls –l** command leads you to believe. For more information see the "Mapping Remote Users" section of Chapter 10 of the *User's/System Administrator's Guide.*

SEE ALSO

chmod(1).
chown(2), group(4), passwd(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

chroot − change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

The *chroot* command causes the given command to be executed relative to the new root. The meaning of any initial slashes (/) in the path names is changed for the command and any of its child processes to *newroot*. Furthermore, upon execution, the initial working directory is *newroot*.

Notice, however, that if you redirect the output of the command to a file:

chroot newroot command >x

will create the file **x** relative to the original root of the command, not the new one.

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.

This command can be run only by the super-user.

SEE ALSO

cd(1).
chroot(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

BUGS

One should exercise extreme caution when referencing device files in the new root file system.

NAME

chrtbl − generate character classification and conversion tables

SYNOPSIS

chrtbl [file]

DESCRIPTION

The *chrtbl* command creates a character classification table and an upper/lower-case conversion table. The tables are contained in a byte-sized array encoded such that a table lookup can be used to determine the character classification of a character or to convert a character [see *ctype*(3C)]. The size of the array is 257*2 bytes: 257 bytes are required for the 8-bit code set character classification table and 257 bytes for the upper- to lower-case and lower- to upper-case conversion table.

chrtbl reads the user-defined character classification and conversion information from *file* and creates two output files in the current directory. One output file, **ctype.c** (a C-language source file), contains the 257*2-byte array generated from processing the information from *file*. You should review the content of **ctype.c** to verify that the array is set up as you had planned. (In addition, an application program could use **ctype.c**.) The first 257 bytes of the array in **ctype.c** are used for character classification. The characters used for initializing these bytes of the array represent character classifications that are defined in **/usr/include/ctype.h**; for example, **_L** means a character is lower case and **_S|_B** means the character is both a spacing character and a blank. The last 257 bytes of the array are used for character conversion. These bytes of the array are initialized so that characters for which you do not provide conversion information will be converted to themselves. When you do provide conversion information, the first value of the pair is stored where the second one would be stored normally, and vice versa; for example, if you provide **<0x41 0x61>**, then **0x61** is stored where **0x41** would be stored normally, and **0x61** is stored where **0x41** would be stored normally.

The second output file (a data file) contains the same information, but is structured for efficient use by the character classification and conversion routines [see *ctype*(3C)]. The name of this output file is the value of the character classification **chrclass** read in from *file*. This output file must be installed in the **/lib/chrclass** directory under this name by someone who is super-user or a member of group **bin**. This file must be readable by user, group, and other; no other permissions should be set. To use the character classification and conversion tables on this file, set the environmental variable **CHRCLASS** [see *environ*(5)] to the name of this file and export the variable; for example, if the name of this file (and character class) is **xyz**, you should issue the commands: **CHRCLASS=xyz ; export CHRCLASS .**

If no input file is given, or if the argument − is encountered, *chrtbl* reads from the standard input file.

The syntax of *file* allows the user to define the name of the data file created by *chrtbl*, the assignment of characters to character classifications and the relationship between upper- and lower-case letters. The character classifications recognized by *chrtbl* are:

chrclass	name of the data file to be created by *chrtbl*.
isupper	character codes to be classified as upper-case letters.
islower	character codes to be classified as lower-case letters.
isdigit	character codes to be classified as numeric.
isspace	character codes to be classified as a spacing (delimiter) character.
ispunct	character codes to be classified as a punctuation character.
iscntrl	character codes to be classified as a control character.
isblank	character code for the space character.
isxdigit	character codes to be classified as hexadecimal digits.
ul	relationship between upper- and lower-case characters.

Any lines with the number sign (#) in the first column are treated as comments and are ignored. Blank lines are also ignored.

A character can be represented as a hexadecimal or octal constant (for example, the letter **a** can be represented as 0x61 in hexadecimal or 0141 in octal). Hexadecimal and octal constants may be separated by one or more space and tab characters.

The dash character (−) may be used to indicate a range of consecutive numbers. Zero or more space characters may be used for separating the dash character from the numbers.

The backslash character (\) is used for line continuation. Only a carriage return is permitted after the backslash character.

The relationship between upper- and lower-case letters (**ul**) is expressed as ordered pairs of octal or hexadecimal constants: <*upper-case_character lower-case_character*>. These two constants may be separated by one or more space characters. Zero or more space characters may be used for separating the angle brackets (< >) from the numbers.

EXAMPLE

The following is an example of an input file used to create the ASCII code set definition table on a file named **ascii**:

```
chrclass   ascii
isupper    0x41 - 0x5a
islower    0x61 - 0x7a
isdigit    0x30 - 0x39
isspace    0x20 0x9 - 0xd
ispunct    0x21 - 0x2f   0x3a - 0x40  \
           0x5b - 0x60   0x7b - 0x7e
iscntrl    0x0 - 0x1f    0x7f
```

```
isblank     0x20
isxdigit    0x30 - 0x39   0x61 - 0x66 \
            0x41 - 0x46
ul          <0x41 0x61> <0x42 0x62> <0x43 0x63> \
            <0x44 0x64> <0x45 0x65> <0x46 0x66> \
            <0x47 0x67> <0x48 0x68> <0x49 0x69> \
            <0x4a 0x6a> <0x4b 0x6b> <0x4c 0x6c> \
            <0x4d 0x6d> <0x4e 0x6e> <0x4f 0x6f> \
            <0x50 0x70> <0x51 0x71> <0x52 0x72> \
            <0x53 0x73> <0x54 0x74> <0x55 0x75> \
            <0x56 0x76> <0x57 0x77> <0x58 0x78> \
            <0x59 0x79> <0x5a 0x7a>
```

FILES

/lib/chrclass/* — data file containing character classification and conversion tables created by *chrtbl*

/usr/include/ctype.h — header file containing information used by character classification and conversion routines

SEE ALSO

ctype(3C), environ(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

DIAGNOSTICS

The error messages produced by *chrtbl* are intended to be self-explanatory. They indicate errors in the command line or syntactic errors encountered within the input file.

NAME

clear – clear terminal screen

SYNOPSIS

/bin/clear

DESCRIPTION

clear clears your screen if it is possible to do so. It looks in the environment for the terminal type and uses terminfo to figure out how to clear the screen.

NAME

clri – clear inode

SYNOPSIS

/etc/clri special i-number ...

DESCRIPTION

The *clri* command writes nulls on the 64 bytes at offset *i-number* from the start of the inode list. This effectively eliminates the inode at that address. *Special* is the device name on which a file system has been defined. After *clri* is executed, any blocks in the affected file will show up as "not accounted for" when *fsck*(1M) is run against the file system. The inode may be allocated to a new file.

Read and write permission is required on the specified *special* device.

This command is used to remove a file which appears in no directory; that is, to get rid of a file which cannot be removed with the *rm* command.

SEE ALSO

fsck(1M), fsdb(1M), ncheck(1M) rm(1).
fs(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

WARNINGS

If the file is open for writing, *clri* will not work. The file system containing the file should NOT be mounted.

If *clri* is used on the inode number of a file that does appear in a directory, it is imperative to remove the entry in the directory at once, since the inode may be allocated to a new file. The old directory entry, if not removed, continues to point to the same file. This sounds like a link, but does not work like one. Removing the old entry destroys the new file.

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.

NAME

col – filter reverse line-feeds

SYNOPSIS

col [–b] [–f] [–x] [–p]

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** (\017) and **SI** (\016) 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 escape sequences unknown to it that are 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.

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; use **–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

comm − select or reject lines common to two sorted files

SYNOPSIS

comm [− [123]] file1 file2

DESCRIPTION

The *comm* command 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** prints nothing.

SEE ALSO

cmp(1), diff(1), sort(1), uniq(1).

NAME

copy – copy groups of files

SYNOPSIS

copy [*option*] ... *source* ... *dest*

DESCRIPTION

The *copy* command copies the contents of directories to another directory. It is possible to copy whole file systems since directories are made when needed.

If files, directories, or special files do not exist at the destination, then they are created with the same modes and flags as the source. In addition, the super-user may set the user and group ID. The owner and mode are not changed if the destination file exists. Note that there may be more than one source directory. If so, the effect is the same as if the *copy* command had been issued for each source directory with the same destination directory for each copy.

All options must be given as separate arguments, and they may appear in any order. The options are:

–a Asks the user before attempting a copy. If the response does not begin with a "y", then a copy is not done. This option also sets the **–ad** option.

–l Uses links instead whenever they can be used. Otherwise, a copy is done. Note that links are never done for special files or directories.

–n Requires the destination file to be new. If not, then the *copy* command does not change the destination file. The **–n** flag is meaningless for directories. For special files, an **–n** flag is assumed (i.e., the destination of a special file must not exist).

–o If set then every file copied has its owner and group set to those of the source. If not set, then the file's owner is the user who invoked the program.

–m If set, then every file copied has its modification time and access time set to that of the source. If not set, then the modification time is set to the time of the copy.

–r If set, then every directory is recursively examined as it is encountered. If not set, then any directories that are found are ignored.

–ad Asks the user whether an **–r** flag applies when a directory is discovered. If the answer does not begin with a "y", then the directory is ignored.

–v If the verbose option is set, messages are printed that reveal what the program is doing.

source This may be a file, directory, or special file. It must exist. If it is not a directory, then the results of the command are the same as for the *cp* command.

dest The destination must be either a file or directory that is different from the source.

If *source* and *destination* are anything but directories, then *copy* acts just like a *cp* command. If both are directories, then *copy* copies each file into the destination directory according to the flags that have been set.

SEE ALSO

chmod(1), cp(1).

NOTES

Special device files can be copied. When they are copied, any data associated with the specified device is *not* copied.

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. For *mv*, when source parent directories or the target directory is writable and has the sticky bit set, any of the following conditions must be true:

the user must own the file
the user must own the directory
the file must be writable to the user
the user must be the super-user

When the **–f** option is used or if the standard input is not a terminal, no questions are asked and the *mv* or *ln* is done.

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

chmod(1), cpio(1), rm(1).

WARNINGS

ln will not link across file systems. This restriction is necessary because file systems can be added and removed.

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.

NAME

cpio − copy file archives in and out

SYNOPSIS

cpio −o [**acBvV**] [**−C** bufsize] [[**−O** file] [**−M** message]]

cpio −i [**BcdmrtuvVfsSb6k**] [**−C** bufsize] [[**−I** file] [**−M** message]] [pattern ...]

cpio −p [**adlmuvV**] 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 by default.

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 regular expressions given in the filename-generating notation of *sh*(1). In *patterns*, metacharacters ?, *, and [...] match the slash (/) character, and backslash (\) is an escape character. A ! metacharacter means *not*. (For example, the **!abc*** pattern would exclude all files that begin with **abc**.) Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (i.e., select all files). Each *pattern* must be enclosed in double quotes otherwise the name of a file in the current directory is used. 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* . NOTE: If *cpio −i* tries to create a file that already exists and the existing file is the same age or newer, *cpio* will output a warning message and not replace the file. (The **−u** option can be used to unconditionally overwrite the existing file.)

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. Archives of text files created by *cpio* are portable between implementations of UNIX System V.

The meanings of the available options are:

- **−a** Reset *access* times of input files after they have been copied. Access times are not reset for linked files when *cpio −pla* is specified.
- **−b** Reverse the order of the *bytes* within each word. Use only with the **−i** option.
- **−B** Input/output is to be blocked 5,120 bytes to the record. The default buffer size is 512 bytes when this and the **−C** options are not used. (**−B** does not apply to the *pass* option; **−B** is meaningful only with data directed to or from a character-special device, e.g., **/dev/rdsk/f0q15dt**.)

−c Write header information in ASCII *character* form for portability. Always use this option when origin and destination machines are different types.

−C *bufsize*
Input/output is to be blocked *bufsize* bytes to the record, where *bufsize* is replaced by a positive integer. The default buffer size is 512 bytes when this and **−B** options are not used. (**−C** does not apply to the *pass* option; **−C** is meaningful only with data directed to or from a character-special device, e.g., **/dev/rmt/c0s0**.)

−d *directories* are to be created as needed.

−f Copy in all *files* except those in *patterns*. (See the paragraph on *cpio −i* for a description of *patterns*.)

−I *file* Read the contents of *file* as input. If *file* is a character-special device, when the first medium is full, replace the medium and type a carriage return to continue to the next medium. Use only with the **−i** option.

−k Attempt to skip corrupted file headers and I/O errors that may be encountered. If you want to copy files from a medium that is corrupted or out of sequence, this option lets you read only those files with good headers. (For *cpio* archives that contain other *cpio* archives, if an error is encountered, *cpio* may terminate prematurely. *cpio* will find the next good header, which may be one for a smaller archive, and terminate when the smaller archive's trailer is encountered.) Used only with the **−i** option.

−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.

−M *message*
Define a message to use when switching media. When you use the **−O** or **−I** options and specify a character-special device, you can use this option to define the message that is printed when you reach the end of the medium. One **%d** can be placed in the message to print the sequence number of the next medium needed to continue.

−O *file* Direct the output of cpio to *file*. If *file* is a character-special device, when the first medium is full, replace the medium and type a carriage return to continue to the next medium. Use only with the **−o** option.

−r Interactively *rename* files. If the user types a null line, the file is skipped. If the user types a ".", the original pathname will be copied. (Not available with *cpio −p*.)

−s *swap* bytes within each half word. Use only with the **−i** option.

−S *Swap* halfwords within each word. Use only with the **−i** option.

−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)].

−V *Special*Verbose: print a dot for each file seen. Useful to assure the user that *cpio* is working without printing out all file names.

−6 Process an old (i.e., UNIX System *Sixth* Edition format) file. Use only with the **−i** option.

NOTE: *cpio* assumes 4-byte words.

If *cpio* reaches end of medium (end of a diskette for example) when writing to (**−o**) or reading from (**−i**) a character-special device, and **−O** and **−I** are not used, *cpio* will print the message:

If you want to go on, type device/file name when ready.

To continue, you must replace the medium and type the character-special device name (**/dev/rdsk/f0q15dt** for example) and a carriage return. You may want to continue by directing *cpio* to use a different device. For example, if you have two floppy drives, you may want to switch between them so *cpio* can proceed while you are changing the floppies. (A carriage return alone causes the *cpio* process to exit.)

EXAMPLES

The following examples show three uses of *cpio.*

When standard input is directed through a pipe to **cpio −o**, it groups the files so they can be directed (>) to a single file (*../newfile*). The **−c** option insures that the file will be portable to other machines. Instead of *ls*(1), you could use *find*(1), *echo*(1), *cat*(1), etc., to pipe a list of names to *cpio*. You could direct the output to a device instead of a file.

ls **|** cpio −oc > *../newfile*

cpio −i uses the output file of **cpio −o** (directed through a pipe with **cat** in the example), extracts those files that match the patterns (**memo/a1**, **memo/b***), creates directories below the current directory as needed (**−d** option), and places the files in the appropriate directories. The **−c** option is used when the file is created with a portable header. If no patterns were given, all files from *newfile* would be placed in the directory.

cat newfile **|** cpio −icd "*memo/a1*" "*memo/b**"

cpio −p takes the file names piped to it and copies or links (**−l** option) those files to another directory on your machine (*newdir* in the example). The **−d** options says to create directories as needed. The **−m** option says retain the modification time. [It is important to use the **−depth** option of *find*(1) to generate path names for *cpio*. This eliminates problems *cpio* could have trying to create files under read-only directories.]

find . −depth −print **|** cpio −pdlmv *newdir*

SEE ALSO

cat(1), echo(1), find(1), ls(1), tar(1).

cpio(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NOTES

1) Path names are restricted to 256 characters.
2) Only the super-user can copy special files.
3) Blocks are reported in 512-byte quantities.
4) If a file has 000 permissions, contains more than 0 characters of data, and the user is not root, the file will not be saved or restored.

NAME

crash – examine system images

SYNOPSIS

/etc/crash [**–d** dumpfile] [**–n** namelist] [**–w** outputfile]

DESCRIPTION

The *crash* command is used to examine the system memory image of a live or a crashed system by formatting and printing control structures, tables, and other information. Command line arguments to *crash* are *dumpfile, namelist*, and *outputfile.*

Dumpfile is the file containing the system memory image. The default *dumpfile* is */dev/mem.*

The text file *namelist* contains the symbol table information needed for symbolic access to the system memory image to be examined. The default *namelist* is */unix.* If a system image from another machine is to be examined, the corresponding text file must be copied from that machine.

When the *crash* command is invoked, a session is initiated. The output from a *crash* session is directed to *outputfile.* The default *outputfile* is the standard output.

Input during a *crash* session is of the form:

function [argument ...]

where *function* is one of the *crash* functions described in the FUNCTIONS section of this manual page, and *arguments* are qualifying data that indicate which items of the system image are to be printed.

The default for process-related items is the current process for a running system and the process that was running at the time of the crash for a crashed system. If the contents of a table are being dumped, the default is all active table entries.

The following function options are available to *crash* functions wherever they are semantically valid.

–e Display every entry in a table.

–f Display the full structure.

–p Interpret all address arguments in the command line as *physical* addresses.

–s *process*
Specify a process slot other than the default.

–w *file* Redirect the output of a function to *file.*

Note that if the **–p** option is used, all address and symbol arguments explicitly entered on the command line will be interpreted as physical addresses. If they are not physical addresses, results will be inconsistent.

The functions *mode*, *defproc*, and *redirect* correspond to the function options **–p**, **–s**, and **–w**. The *mode* function may be used to set the address translation mode to physical or virtual for all subsequently entered functions; *defproc* sets the value of the process slot argument for subsequent functions; and *redirect* redirects all subsequent output.

Output from *crash* functions may be piped to another program in the following way:

function [argument ...]!shell_command

For example,

mount ! grep rw

will write all mount table entries with an *rw* flag to the standard output. The redirection option (**−w**) cannot be used with this feature.

Depending on the context of the function, numeric arguments will be assumed to be in a specific radix. Counts are assumed to be decimal. Addresses are always hexadecimal. Table slot arguments are always decimal. Table slot arguments larger than the size of the function table will not be interpreted correctly. Use the *findslot* command to translate from an address to a table slot number. Default bases on all arguments may be overridden. The C conventions for designating the bases of numbers are recognized. A number that is usually interpreted as decimal will be interpreted as hexadecimal if it is preceded by **0x** and as octal if it is preceded by **0**. Decimal override is designated by **0d**, and binary by **0b**.

Aliases for functions may be any uniquely identifiable initial substring of the function name. Traditional aliases of one letter, such as **p** for *proc*, remain valid.

Many functions accept different forms of entry for the same argument. Requests for table information will accept a table entry number or a range. A range of slot numbers may be specified in the form *a−b* where *a* and *b* are decimal numbers. An expression consists of two operands and an operator. An operand may be an address, a symbol, or a number; the operator may be +, −, *, /, &, or | . An operand which is a number should be preceded by a radix prefix if it is not a decimal number (**0** for octal, **0x** for hexadecimal, **0b** for binary). The expression must be enclosed in parentheses ( ). Other functions will accept any of these argument forms that are meaningful.

Two abbreviated arguments to *crash* functions are used throughout. Both accept data entered in several forms. They may be expanded into the following:

table_entry = table entry| range

start_addr = address| symbol| expression

FUNCTIONS

? [**−w** file]
List available functions.

!cmd Escape to the shell to execute a command.

adv [**−e**] [**−w** file] [[**−p**]table_entry ...]
Print the advertised table.

base [**−w** file] number ...
Print *number* in binary, octal, decimal, and hexadecimal. A number in a radix other than decimal should be preceded by a prefix that indicates its radix as follows: **0x**, hexadecimal; **0**, octal; and **0b**, binary.

buffer [**−w** file] [−format] bufferslot

or

buffer [**−w** file] [−format] [**−p**] start_addr

Alias: **b**.

Print the contents of a buffer in the designated format. The following format designations are recognized: **−b**, byte: **−c**, character; **−d**, decimal; **−x**, hexadecimal; **−o**, octal; **−r**, directory; and **−i**, inode. If no format is given, the previous format is used. The default format at the beginning of a *crash* session is hexadecimal.

bufhdr [**−f**] [**−w** file] [[**−p**]table_entry...]

Alias: **buf**.

Print system buffer headers.

callout [**−w** file]

Alias: **c**.

Print the callout table.

dballoc [**−w** file] [class ...]

Print the dballoc table. If a class is entered, only data block allocation information for that class will be printed.

dbfree [**−w** file] [class ...]

Print free streams data block headers. If a class is entered, only data block headers for the class specified will be printed.

dblock [**−e**] [**−w** file] [**−c** class...]

or

dblock [**−e**] [**−w** file] [[**−p**] table_entry...]

Print allocated streams data block headers. If the class option (**−c**) is used, only data block headers for the class specified will be printed.

defproc [**−w** file] [**−c**]

or

defproc [**−w** file] [slot]

Set the value of the process slot argument. The process slot argument may be set to the current slot number (**−c**) or the slot number may be specified. If no argument is entered, the value of the previously set slot number is printed. At the start of a *crash* session, the process slot is set to the current process.

dis [**−w** file] [**−a**] start_addr [count]

Disassemble from the start address for *count* instructions. The default count is 1. The absolute option (**−a**) specifies a non-symbolic disassembly.

ds [**−w** file] virtual_address ...

Print the data symbol whose address is closest to, but not greater than, the address entered.

file [**−e**] [**−w** file] [[**−p**]table_entry...]

Alias: **f**.

Print the file table.

findaddr [−**w** file] table slot
Print the address of *slot* in *table*. Only tables available to the *size* function are available to *findaddr*.

findslot [−**w** file] virtual_address ...
Print the table, entry slot number, and offset for the address entered. Only tables available to the *size* function are available to *findslot*.

fs [−**w** file] [[−**p**] table_entry ...]
Print the file system information table.

gdp [−**e**] [−**f**] [−**w** file] [[−**p**] table_entry ...]
Print the gift descriptor protocol table.

gdt [−**e**] [−**w** file] [[−**p**] table_entry ...]
Print the global descriptor table.

help [−**w** file] function ...
Print a description of the named function, including syntax and aliases.

idt [−**e**] [−**w** file] [[−**p**] table_entry ...]
Print the interrupt descriptor table.

inode [−**e**] [−**f**] [−**w** file] [[−**p**] table_entry ...]
Alias: **i**.
Print the inode table, including file system switch information.

kfp [−**w** file] [value]
Print the frame pointer for the start of a kernel stack trace. If the value argument is supplied, the kfp is set to that value.

lck [−**e**] [−**w** file] [[−**p**] table_entry ...]
Alias: **l**.
Print record-locking information. If the −**e** option is used or table address arguments are given, the record lock list is printed. If no argument is entered, information on locks relative to inodes is printed.

ldt [−**e**] [−**w** file] [**-s** process] [[−**p**] table_entry ...]
Print the local descriptor table for the given process, or for the current process if none is given.

linkblk [−**e**] [−**w** file] [[−**p**] table_entry ...]
Print the linkblk table.

map [−**w** file] mapname ...
Print the map structure of *mapname*.

mbfree [−**w** file]
Print free streams message block headers.

mblock [−**e**] [−**w** filename] [[−**p**] table_entry ...]
Print allocated streams message block headers.

mode [−**w** file] [mode]
Set address translation of arguments to virtual (**v**) or physical (**p**) mode. If no mode argument is given, the current mode is printed. At the start of a *crash* session, the mode is virtual.

mount [**−e**] [**−w** file] [[**−p**] table_entry ...]
Alias: **m**.
Print the mount table.

nm [**−w** file] symbol ...
Print value and type for the given symbol.

od [**−p**] [**−w** file] [−format] [−mode] [**−s** process] start_addr [count]
Alias: **rd**.
Print *count* values starting at the start address in one of the following formats: character (**−c**), decimal (**−d**), hexadecimal (**−x**), octal (**−o**), ASCII (**−a**), or hexadecimal/character (**−h**), and one of the following modes: long (**−l**), short (**−t**), or byte (**−b**). The default mode for character and ASCII formats is byte; the default mode for decimal, hexadecimal, and octal formats is long. The format **−h** prints both hexadecimal and character representations of the addresses dumped; no mode needs to be specified. When format or mode is omitted, the previous value is used. At the start of a *crash* session, the format is hexadecimal and the mode is long. If no count is entered, 1 is assumed.

panic
Print the latest system notices, warnings, and panic messages from the limited circular buffer kept in memory.

pcb [**−w** file] [process]
Print the process control block (TSS) for the given process. If no arguments are given, the active TSS for the current process is printed.

pdt [**−e**] [**−w** file] [**−s** process] [**−p**] start_addr [count]
The page descriptor table of the designated memory *section* and *segment* is printed. Alternatively, the page descriptor table starting at the start address for *count* entries is printed. If no count is entered, 1 is assumed.

pfdat [**−e**] [**−w** file] [[**−p**] table_entry ...]
Print the pfdata table.

proc [**−e**] [**−f**] [**−w** file] [[**−p**] table_entry ... #procid ...]
or
proc [**−f**] [**−w** file] [**−r**]
Alias: **p**.
Print the process table. Process table information may be specified in two ways. First, any mixture of table entries and process ids may be entered. Each process id must be preceded by a **#**. Alternatively, process table information for executable processes may be specified with the executable option (**−r**). The full option (**−f**) details most of the information in the process table as well as the region table for that process.

qrun [**−w** file]
Print the list of scheduled streams queues.

queue [**−e**] [**−w** file] [[**−p**]table_entry ...]
Print streams queues.

quit Alias: **q**.
Terminate the *crash* session.

rcvd [**−e**] [**−f**] [**−w** file] [[**−p**]table_entry ...]
Print the receive descriptor table.

redirect [**−w** file] [**−c**]

or

redirect [**−w** file] [file]
Used with a file name, redirects output of a *crash* session to the named file. If no argument is given, the file name to which output is being redirected is printed. Alternatively, the close option (**−c**) closes the previously set file and redirects output to the standard output.

region [**−e**] [**−w** file] [[**−p**]table_entry ...]
Print the region table.

sdt [**−e**] [**−w** file] [**−s** process] section

or

sdt [**−e**] [**−w** file] [**−s** process] [**−p**] start_addr [count]
The segment descriptor table for the current process is printed.

search [**−p**] [**−w** file] [**−m** mask] [**−s** process]
pattern start_addr count
Print the long words in memory that match *pattern*, beginning at the start address for *count* long words. The mask is anded (&) with each memory word and the result compared against the pattern. The mask defaults to 0xffffffff.

size [**−w** file] [**−x**] [structure_name ...]
Print the size of the designated structure. The (**−x**) option prints the size in hexadecimal. If no argument is given, a list of the structure names for which sizes are available is printed.

sndd [**−e**] [**−f**] [**−w** file] [[**−p**]table_entry ...]
Print the send descriptor table.

srmount [**−e**] [**−w** file] [[**−p**]table_entry ...]
Print the server mount table.

stack [**−w** file] [process]
Alias: **s**.
Dump stack. If no arguments are entered, the kernel stack for the current process is printed. The interrupt stack and the stack for the current process are not available on a running system.

stat [**−w** file]
Print system statistics.

stream [**−e**] [**−f**] [**−w** file] [[**−p**]table_entry ...]
Print the streams table.

strstat [**−w** file]
Print streams statistics.

trace [**−w** file] [**−r**] [process]
Alias: **t**.
Print kernel stack trace. The kfp value is used with the **−r** option.

ts [**−w** file] virtual_address ...
Print closest text symbol to the designated address.

tty [**−e**] [**−f**] [**−w** file] [**−t** type [[**−p**] table_entry ...]]
Valid types: **co, c1, c2** (console, com1, com2).
Print the tty table. If no arguments are given, the tty table for the console is printed. If the **−t** option is used, the table for the single tty type specified is printed. If no argument follows the type option, all entries in the table are printed. A single tty entry may be specified from the start address.

user [**−f**] [**−w** file] [process]
Alias: **u**.
Print the ublock for the designated process.

var [**−w** file]
Alias: **v**.
Print the tunable system parameters.

vtop [**−w** file] [**−s** process] start_addr ...
Print the physical address translation of the virtual start address.

FILES

/dev/mem system image of currently running system

NAME

cron – clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

The *cron* command executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in *crontab* files in the directory **/usr/spool/cron/crontabs**. Users can submit their own *crontab* file via the *crontab*(1) command. Commands which are to be executed only once may be submitted via the *at*(1) command.

The *cron* command only examines *crontab* files and *at* command files during process initialization and when a file changes via *crontab* or *at*. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Since *cron* never exits, it should be executed only once. This is done routinely through **/etc/rc2.d/S75cron** at system boot time. **/usr/lib/cron/FIFO** is used as a lock file to prevent the execution of more than one *cron*.

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/FIFO	used as a lock file
/usr/lib/cron/log	accounting information
/usr/spool/cron	spool area
/usr/default/cron	

SEE ALSO

at(1), crontab(1), sh(1).

DIAGNOSTICS

To keep a log of all actions taken by *cron* , set CRONLOG=YES in the file **/etc/default/cron**. No logging is done if you set CRONLOG=NO (default setting). Keeping a log is a user-configurable option. *cron* usually creates very large log files.

NAME

crontab — user crontab file

SYNOPSIS

crontab [file]
crontab −r
crontab −l

DESCRIPTION

The *crontab* command 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 **cron.allow** does not exist and **cron.deny** exists but is empty, 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/lbin).**

If you do not redirect the standard output and standard error of your commands, any generated output or errors will be mailed to you.

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

cron(1M), sh(1).

WARNINGS

If you inadvertently enter the *crontab* command with no argument(s), do not attempt to get out with a CTRL-d. This will cause all entries in your *crontab* file to be removed. Instead, exit with a DEL.

NAME

csh – invoke a shell command interpreter that uses C-like syntax

SYNOPSIS

csh [−cefinstvVxX] [*arg* ... **]**

DESCRIPTION

csh is a command language interpreter. It 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 user's **.login** file. 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 user's home directory.

Lexical Structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters **&**, **|**, **;**, **<**, **>**, **(**, and **)** form separate words. If doubled (for example, **&&**, **||**, **<<**, or **>>**) these character pairs form single words. These parser metacharacters may be made part of other words, or you can take away their special meaning by preceding them with a backslash character (\). A new-line character preceded by a \ is equivalent to a blank.

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. The semantics of these quotations are described below. Within pairs of ′ or " characters, a new-line character preceded by a \ gives a true new-line character.

When the shell's input is not from a terminal, the character **#** introduces a comment which continues to the end of the input line. It does not have this special meaning when preceded by \ or when placed inside the quotation marks ‵, ′, 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 provides the input for the next command. Sequences of pipelines may be executed sequentially by separating them with the ; character. A sequence of pipelines may be executed in the background by following it with the **&** character.

Any of the above may be placed in parentheses 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 that looks like:

[1] 1234

indicating that the job that was started asynchronously was job number 1 and it had one (top-level) process, whose process ID was 1234.

If you are running a job and wish to do something else, you may use **^Z** (CTRL-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 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 that 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 terminal option, then background jobs will stop when they try to produce output as they do when they try 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 as **%1**. Just 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**. It is also possible to 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. This is done 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.

Substitutions

The following sections describe the various transformations the shell performs on the input in the order in which they occur.

History Substitutions

History substitutions can be used to reintroduce sequences of words from previous commands, possibly performing modifications on these words. Thus, history substitutions provide a generalization of a *redo* function.

History substitutions begin with the ! character and may begin anywhere in the input stream if a history substitution is not already in progress. This ! may be preceded by a \ to prevent its special meaning; a ! is passed unchanged when it is followed by a blank, tab, new-line character, =, or (. History substitutions also occur when an input line begins with ^. This special abbreviation will be described later.

Any input line that contains a history substitution is echoed on the terminal before it is executed as it could have been typed without a history substitution.

Commands input from the terminal that consist of one or more words are saved on the history list, the size of which is controlled by the *history* variable. The previous command is always retained. Commands are numbered sequentially from 1.

For example, 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 in **!-2** (referring to the same event), by a prefix of a command word as in **!d** for event 12 or **!w** 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 a *redo*. The form **!#** references the current command (the one being typed in). It allows a word to be selected from further left in the line, to avoid retyping a long name, as in **!#:1**.

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, and so on. The basic word designators are as follows:

Designator	**Description**
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
−*y*	Abbreviates 0−*y*
*	Abbreviates ^−$, or nothing if only one word in event
x *	Abbreviates *x*−$
x−	Like *x* * but omitting word $

The colon separating the event specification from the word designator can be omitted if the argument selector begins with a ↑, $, *, −, or %. After the optional word designator, a sequence of modifiers can be placed, each preceded by a :. The following modifiers are defined:

Modifier	**Description**
h	Removes a trailing path name component.
r	Removes a trailing *.xxx* component.
s/*l*/*r*/	Substitutes *l* for *r*.
t	Removes all leading path name components.
&	Repeats the previous substitution.
g	Applies the change globally, prefixing the above.
p	Prints the new command but does not execute it.
q	Quotes the substituted words, preventing substitutions.
x	Like **q**, but breaks into words at blanks, tabs, and new-line characters.

Unless preceded by a **g**, the modification is applied only to the first modifiable word. In any case, it is an error for no word to be applicable.

The left 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 slash (/); a backslash (\) quotes the delimiter into the *l* and *r* strings. The character **&** in the right-hand side is replaced by the text from the left-hand side. A \ quotes **&** also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in **!?***s***?**. The trailing delimiter in the substitution may be omitted if a new-line character follows immediately, as may the trailing ? in a contextual scan.

A history reference may be given without an event 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 nonblank 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, *^lb^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 that follow. Thus, after *ls* **−ld** *~paul* we might do **!{l}***a* to do *ls* **−ld** *~paula,* while *!la* would look for a command starting with *la*.

Quotations With Single Quotes (') and Double Quotes (")

The quotation of strings by ′ and " can be used to prevent all or some of the remaining substitutions. Strings enclosed in ′ are prevented from any further interpretation. Strings enclosed in " are variable, and command expansion may occur.

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 that can be established, displayed, and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands, and the first word of each command, 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 command 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**. 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. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus, you can alias *print* **′pr \!* | lpr′** to make a command that paginates its arguments to the line printer.

Variable Substitution

The shell maintains a set of variables, each of which has as value 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* commands. 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 that causes command input to be echoed. The setting of this variable results from the **−v** command line option.

Other operations treat variables numerically. The at-sign (@) command permits numeric calculations to be performed and the result assigned to a variable. However, variable values are 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 dollar sign (**$**) characters. This expansion can be prevented by preceding the dollar sign with a backslash (\) except within double quotation marks ("), in which case it *always* occurs, and within single quotation marks (′) where it *never* occurs. Strings quoted by back quotation marks (\`) are interpreted later (see *Command Substitution* below) so dollar sign substitution does not occur there until later, if at all. A dollar sign is passed unchanged if followed by a blank, tab, or end-of-line.

Input and output redirections are recognized before variable expansion and are expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in double quotation marks or given the **:q** modifier, the results of variable substitution may eventually be command and file name substituted. Within double quotation marks (") a variable whose value consists of multiple words expands to a portion of a single word, with the words of the variable's value separated by blanks. When the **:q** modifier is applied to a substitution, the variable expands to multiple words with each word separated by a blank and quoted to prevent later command or file name substitution.

The following sequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable that 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, digits, and underscores.

If *name* is not a shell variable, but is set in the environment, then that value is returned. However, modifiers and the other forms shown in the following list 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 **$** substitution and may consist of a single number or two numbers separated by a **−**. The first word of a variable value is numbered **1**. If the first number of a range is omitted, it defaults to **1**. If the last member of a range is omitted, it defaults to **$#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. Only one : modifier is allowed on each **$** expansion.

The following substitutions may not be modified with : modifiers:

$?*name*
${?*name*}

Substitutes the string 1 if name is set and 0 if it is not.

$?0 Substitutes 1 if the current input file name is known and 0 if it is not.

$$ Substitutes the (decimal) process number of the (parent) shell.

Command and File Name Substitution

Command and file name substitution are applied selectively to the arguments of built-in commands. This means that portions of expressions that are not evaluated are not subjected to these expansions. For commands that 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 back quotation marks. The output from such a command is normally broken into separate words at blanks, tabs, and new-lines, with null words being discarded. This text then replaces the original string. Within double quotation marks, only new-line characters force new words; blanks and tabs are preserved.

In any case, the single final new-line character 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 displays a complete line.

File Name Substitution

If a word contains any of the characters *, ?, [, or {, or begins with the character ~, then that word is a candidate for file name substitution, also known as "globbing." This word is then regarded as a pattern and is replaced with an alphabetically sorted list of file names that match the pattern. In a list of words specifying file name 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 file names, the character . at the beginning of a file name 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 file name is used to refer to home directories. Standing alone it expands to the invoker's 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 the user's 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 /, or does not appear at the beginning of a word, it is left unchanged.

The metanotation **a{b,c,d}e** is a shorthand for **abe 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/s1/{oldls,ls}.c** expands to **/usr/source/s1/oldls.c /usr/source/s1/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 unchanged.

Input/Output

The standard input and standard output of a command may be redirected with the following syntax:

< *name*

Opens file *name* (which is first variable, command, and file name expanded) as the standard input.

<< *word*

Reads the shell input up to a line which is identical to *word*. *word* is not subjected to variable, file name, or command substitution, and each input line is compared to *word* before any substitutions are made on this input line. Unless a quoting backslash, double or single quotation mark, or a back quotation mark appears in *word*, variable and command substitution is performed on the intervening lines, allowing \ to quote $, \, and `. Commands that are substituted have all blanks, tabs, and new-line characters preserved except for the final new-line character, which is dropped. The resulting 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 its previous contents are lost.

If the variable *noclobber* is set, then the file must not already exist or it must 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 to 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 file names 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, it is similar to >.

If a command is run detached (followed by **&**), then the default standard input for the command is the empty file **/dev/null**. Otherwise, the 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.

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 later) 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, with the following operators forming groups at the same level:

== and !=
<=, >=, <, and >
<< and >>
+ and −
* / and %

The == and != operators compare their arguments as strings; all others operate on numbers. Strings that 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; they should be surrounded by spaces except when adjacent to components of expressions which are syntactically significant to the parser **[& | < > ( )]**.

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 the following characters:

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 file name-expanded, 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 that can be used to control command files (shell scripts) and, in limited but useful ways, 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 command line.

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** commands will succeed on nonseekable 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. The following list describes the syntax and function of the built-in commands:

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 file name substituted. *name* is not allowed to be **alias** or **unalias**.

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* statement. The remaining commands on the current line are executed. Multilevel 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*
Changes the shell's working directory to directory *name*. If no argument is given, then it changes 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
Continues 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*
The specified words are written to the shell's standard output. A **\c** causes the echo to complete without printing a new-line character. A **\n** in *wordlist* causes a new-line character to be printed. Otherwise, the words are echoed, separated by spaces.

else
end
endif
endsw
See the following descriptions of the *foreach*, *if*, *switch*, and *while* statements.

exec *command*

The specified command 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).

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* may be used 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.

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 file name-expand a list of words.

goto *word*

The specified *word* is file name-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.

history

Displays the history event list.

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 command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is *not* executed.

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 an *else*.)

jobs
jobs −l
Lists the active jobs; given the **−l** options lists process IDs 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/inlude/signal.h**, stripped of the prefix "SIG"). The signal names are listed by "kill −l." There is no default; saying only "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*
limit −h
limit −h *resource*
limit −h *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. If the **−h** flag is given, the hard limits are used instead of the current limits. The hard limits impose a ceiling on the values of the current limits. Only the superuser may raise the hard limits, but a user may lower or raise the current limits within the legal range.

Resources controllable currently include *cputime* (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 *sbrk*(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-size* may be given as a (floating point or integer) number followed by a scale factor. For all limits other than *cputime* the default scale is "k" or "kilobytes" (1024 bytes); a scale factor of "m" or "megabytes" may also be used. For *cputime* 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.

logout

Terminates a login shell. The only way to log out if *ignoreeof* is set.

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 superuser may specify negative niceness by using "nice −number ...." The command is always executed in a subshell, 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. Unless the shell is running detached, *nohup* has no effect. All processes detached with **&** are automatically run with *nohup.* (Thus *nohup* is not really needed.)

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*

Controls the action of the shell on interrupts. The first form restores the default action of the shell on interrupts: 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.

rehash

Causes the internal hash table of the directories' contents 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 be necessary only 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 that 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* 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 file name-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 the environment variable *name* to be *value*, a single string. Useful environment variables are TERM, the type of your terminal, and SHELL, the shell you are using.

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 which is 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* that is first command- and file name-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 all access except no write 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 match the *unalias* pattern.

unhash

Use of the internal hash table to speed location of executed programs is disabled.

unlimit
unlimit *resource*
unlimit **−h**
unlimit **−h** *resource*

Removes the limitation on *resource*. If no *resource* is specified, then all *resource* limitations are removed. If **−h** is given, the corresponding hard limits are removed. Only the superuser may do this.

unset *pattern*

All variables whose names match the specified *pattern* are removed. Thus, all variables are removed by *unset* *; this has noticeably undesirable side-effects. It is not an error for nothing to be *unset*.

wait

All child processes are waited for. It the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and process numbers of all children known to be outstanding.

while(*expr*)

...

end

While the specified expression evaluates nonzero, 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* argument of *name*. Both *name* and its *index* component must already exist.

Assignment operators, such as *= and +=, are available as in C. The space separating the name from the assignment operator is optional. Spaces are mandatory in separating components of *expr* which would otherwise be single words.

Special postfix ++ and − − operators increment and decrement *name* respectively, i.e., **@ i++**.

Predefined Variables

The following variables have special meaning to the shell. Of these, *argv*, *child*, *home*, *path*, *prompt*, *shell*, and *status* are always set by the shell. Except for *child* and *status*, this setting occurs only at initialization; these variables will not then be modified unless done explicitly by the user.

Variable	**Description**
argv	Set to the arguments of the shell; from this variable, positional parameters are substituted, i.e., **$1** is replaced by **$argv**[1].

cdpath Gives a list of alternate directories searched to find subdirectories in *cd* commands.

child The process number printed when the last command was forked with **&**. This variable is *unset* when this process terminates.

echo Set when the **−x** command line option is given. Causes each command and its arguments to be echoed just before it is executed. For nonbuilt-in commands, all expansions occur before echoing. Built-in commands are echoed before command and file name substitution since these substitutions are then done selectively.

histchars Can be assigned a two-character string. The first character is used as a history character in place of **!**; the second character is used in place of the **^** substitution mechanism. For example, *set histchars = ,;* will cause the history characters to be comma and semicolon.

history Can be given a numeric value to control the size of the history list. Any command that has been referenced in this many events will not be discarded. A *history* that is too large may run the shell out of memory. The last executed command is always saved on the history list.

home The home directory of the user, initialized from the environment. The file name expansion of **~** refers to this variable.

ignoreeof If set, the shell ignores the end-of-file from input devices that are terminals. This prevents a shell from accidentally being terminated by typing a CTRL-D.

mail The files where the shell checks for mail. This is done after each command completion results in a prompt, if a specified interval has elapsed. The shell sends the message "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 *mail* 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 sends the message "New mail in *name*" when there is mail in the file *name*.

noclobber Restrictions are placed on output redirection to insure that files are not accidentally destroyed and that **>>** redirections refer to existing files.

noglob If set, file name expansion is inhibited. This is most useful in shell scripts that are not dealing with file names or after a list of file names has been obtained and further expansions are not desirable.

nonomatch If set, it is not an error for a file name 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 *path* variable, then only full path names will execute. The usual search path is **/bin**, **/usr/bin**, and ., but this may vary from system to system. For the superuser, the default search path is **/etc**, **/bin** and **/usr/bin**. A shell that is given neither the **−c** nor the **−t** option will normally hash the contents of the directories in the *path* variable after reading **.cshrc** and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the *rehash*, or the commands may not be found.

prompt The string that 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. The default is % or # for the superuser.

shell The file in which the shell resides. This is used in forking shells to interpret files that have execute bits set but are not executable by the system (see the section *Nonbuilt-In Command Execution* below.) *shell* is 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. Abnormal termination results in a core dump. Built-in commands that fail return exit status 1; all other built-in commands set status 0.

time Controls automatic timing of commands. If set, then any command that takes more than this many CPU seconds will cause a line giving user, system, and real times and a utilization percentage (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.

The shell copies the environment variable PATH into the variable *path* and copies the value back into the environment whenever *path* is set. Thus, 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.

Nonbuilt-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 **−t** 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 **−t** 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 *cd* 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:

Flag	**Description**
−c	Reads commands from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.
−e	Causes the shell to exit if any invoked command terminates abnormally or yields a nonzero exit status.
−f	Lets the shell start faster because it will neither search for nor execute commands from the file **.cshrc** in the user's home directory.
−i	Makes the shell interactive. The shell prompts for its top-level input even if it appears not to be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.
−n	Causes commands to be parsed but not executed. This may aid in syntactic checking of shell scripts.

−s Causes command input to be taken from the standard input.

−t Reads and executes a single line of input. A backslash (\) can be used to escape the new-line character 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 Causes the *echo* variable to be set even before **.cshrc** is executed.

After processing of flag arguments, if arguments remain but none of the **−c**, **−i**, **−s**, or **−t** options were 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 on a typical system most shell scripts are written for the standard shell (see *sh*(1)), the C shell will execute such a standard shell if the first character of a script is not a **#**, i.e., if the script does not start with a comment. Remaining arguments initialize the variable *argv*.

Signal Handling

The shell normally ignores *quit* signals. The *interrupt* and *quit* signals are ignored for an invoked command if the command is followed by **&**; otherwise, the signals have the values that the shell inherited from its parent. The shell's handling of interrupts 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**.

NEW ENVIRONMENT VARIABLES

The new environment variable described in this section has been added to the C shell. The C shell will behave normally for those users who do not set **DOSPATH**. Users who wish to be able to execute MS-DOS (DOS) programs directly from the C shell, that is, bypassing the normal DOS bootup that occurs when running *vpix*, should set **DOSPATH** to include those directories in **PATH** that contain DOS executables.

DOSPATH is a string with the same format as **PATH**; it contains a subset of the list of directories from **PATH**. When searching a directory in **PATH** for a program, the C shell determines whether that directory is also in **DOSPATH**. If it is not, the C shell acts as usual. If it is, the C shell looks first for the command with the suffix **.com**, then **.exe**, then **.bat**, and finally, for the command without any suffix. Whenever the result of a path search gives a file with one of these DOS suffixes, the shell runs the *vpix* program via a standard search path and adds arguments **−c** and the full path name of the DOS program (including the suffix).

For example, if **PATH** is set to **:/bin:/usr/bin**, **DOSPATH** is set to **.**, the current directory is **/usr/john/dosbin**, and there is a DOS program named *abc.comi* in the current directory, then typing *abc* to the C shell will cause the command *vpix* **−c /usr/john/dosbin/abc.com** to be executed, which will run the DOS program *abc.com* without the normal *vpix* DOS bootup.

FILES

~/.cshrc	Read by each shell at the beginning of execution
~/.login	Read by login shell after **.cshrc** at login
~/.logout	Read by login shell at logout
/bin/sh	Shell for scripts not starting with a **#! /bin/csh**
/tmp/sh*	Temporary file for <<
/dev/null	Source of empty file
/etc/passwd	Source of home directories for ~*name*
/etc/default/.cshrc	Default file of automatically invoked commands

SEE ALSO

umask(1), wait(1).
access(2), exec(2), fork(2), pipe(2), signal(2), a.out(4), environ(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

CREDIT

This utility was developed at the University of California at Berkeley and is used with permission.

NOTES

Words can be no longer than 512 characters. The number of arguments to a command which involves file name expansion is limited to 1/6 of the number of characters allowed in an argument list, which is 5120 less the characters in the environment. Also, 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.

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.

Built-in control structure commands like *foreach* and *while* cannot be used with the pipe symbol (|), ampersand (**&**), or semicolon (;).

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 parentheses to force it to a subshell; i.e., "(a;b;c)."

Commands within loops prompted for by **?** are not placed in the *history* list.

It is not possible to use the colon (:) modifiers on the output of command substitutions.

csh attempts to import and export the PATH variable for use with regular shell scripts. This only works for simple cases, where the PATH contains no command characters.

This version of *csh* does not support or use the process control features of the 4th Berkeley Distribution.

You can modify the list of commands that *csh* automatically invokes by editing the **/etc/default/.cshrc** file. For example, if you want to automatically assign the **alias** *h* to the **history** command, add the following line to the **/etc/default/.cshrc** file using the computer editor of your choice:

alias history h

NAME

csplit – context split

SYNOPSIS

csplit [–s] [–k] [–f prefix **]** file arg1 [... argn]

DESCRIPTION

The *csplit* command reads *file* and separates it into n+1 sections, defined by the arguments *arg1... argn*. By default the sections are placed in xx00 ... xx*n* (*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 *prefix***00** ... *prefixn*. The default is **xx00** ... **xx***n*.

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 user's 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).
regexp(5) in the *INTERACTIVE SDS Guide and Programmer's Reference 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.

NAME

ct – spawn getty to a remote terminal

SYNOPSIS

ct [–wn] [–xn] [–h] [–v] [–sspeed] telno ...

DESCRIPTION

The *ct* command dials the telephone 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. (The set of legal characters for *telno* is 0 through 9, –, =, *, and #. The maximum length *telno* is 31 characters). If more than one telephone number is specified, the *ct* command 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/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 **–w***n* option, where *n* is the maximum number of minutes that *ct* is to wait for a line.

The **–x***n* option is used for debugging; it produces a detailed output of the program execution on stderr. The debugging level, *n*, is a single digit; **–x9** is the most useful value.

Normally, *ct* will hang up the current line, so the line can answer the incoming call. The **–h** option will prevent this action. The **–h** option will also wait for the termination of the specified *ct* process before returning control to the user's terminal. 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 1200.

After the user on the destination terminal logs out, there are two things that could occur depending on what type of getty is on the line (*getty* or *uugetty*). For the first case, *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. In the second case, there is already a getty (*uugetty*) on the line, so the **login:** message will appear.

To log out properly, the user must type **control D**.

Of course, the destination terminal must be attached to a modem that can answer the telephone.

FILES

/usr/lib/uucp/Devices
/usr/adm/ctlog

SEE ALSO

cu(1C), getty(1M), login(1), uucp(1C), uugetty(1M).

BUGS

For a shared port, one used for both dial-in and dial-out, the *uugetty* program running on the line must have the **−r** option specified [see *uugetty*(1M)].

NAME

cu — call another UNIX system

SYNOPSIS

cu [**—s**speed] [**—l**line] [**—h**] [**—t**] [**—d**] [**—o** | **—e**] [**—n**] telno
cu [**—s** speed] [**—h**] [**—d**] [**—o** | **—e**] **—l** line
cu [**—h**] [**—d**] [**—o** | **—e**] systemname

DESCRIPTION

The *cu* command calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of ASCII files.

The *cu* command accepts the following options and arguments:

—s*speed* — Specifies the transmission speed (300, 1200, 2400, 4800, 9600); The default value is "Any" speed which will depend on the order of the lines in the **/usr/lib/uucp/Devices** file. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

—l*line* — Specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place 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 Devices file. When the **—l** and **—s** options are both used together, cu will search the 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/tty***ab*) in which case a telephone number (*telno*) is not required. The specified device need not be in the **/dev** directory. If the specified device is associated with an auto dialer, a telephone number must be provided. Use of this option with *systemname* rather than *telno* will not give the desired result (see *systemname* below).

—h — Emulates local echo, supporting calls to other computer systems which expect terminals to be set to half-duplex mode.

—t — Used to dial 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.

—o — Designates that odd parity is to be generated for data sent to the remote system.

—e — Designates that even parity is to be generated for data sent to the remote system.

—n — For added security, will prompt the user to provide the telephone number to be dialed 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 placed appropriately for delays of 4 seconds.

systemname A *uucp* system name may be used rather than a telephone number; in this case, *cu* will obtain an appropriate direct line or telephone number from **/usr/lib/uucp/Systems**. Note: the *systemname* option should not be used in conjunction with the **−l** and **−s** options as *cu* will connect to the first available line for the system name specified, ignoring the requested line and speed.

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 system so the buffer is not overrun. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following user-initiated commands:

~. terminate the conversation.

~! escape to an interactive shell on the local system.

~!*cmd*... run *cmd* on the local system (via **sh −c**).

~$*cmd*... run *cmd* locally and send its output to the remote system.

~%**cd** change the directory on the local system. Note: ~**!cd** will cause the command to be run by a sub-shell, probably not what was intended.

~%**take** *from* [*to*] copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.

~%**put** *from* [*to*] copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places.

For both ~%**take** and **put** commands, as each block of the file is transferred, consecutive single digits are printed to the terminal.

~~ *line* send the line ~ *line* to the remote system.

~%**break** transmit a **BREAK** to the remote system (which can also be specified as ~%**b**).

~%**debug** toggles the **−d** debugging option on or off (which can also be specified as ~%**d**).

~**t** prints the values of the termio structure variables for the user's terminal (useful for debugging).

~l prints the values of the termio structure variables for the remote communication line (useful for debugging).

~%nostop 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. Internally the program accomplishes this by initiating an output diversion to a file when a line from the remote begins with ~.

Data from the remote is diverted (or appended, if >> is used) to *file* on the local system. The trailing ~> marks the end of 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 control characters 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, *tabs* mode [see *stty(1)*] should be set on the remote system if tabs are to be copied without expansion to spaces.

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 ~~. Executing a tilde command reminds the user of the local system uname. For example, *uname* can be executed on Z, X, and Y as follows:

```
uname
Z
~[X]!uname
X
~~[Y]!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 telephone number is 9 201 555 1212 using 1200 baud (where dialtone is expected after the 9):

    cu −s1200 9=12015551212

If the speed is not specified, "Any" is the default value.

To log in to a system connected by a direct line, enter:

    cu −l /dev/ttyXX

or

    cu −l ttyXX

To dial a system with the specific line and a specific speed, enter:

    cu −s1200 −l ttyXX

To dial a system using a specific line associated with an auto dialer, enter:

cu −l culXX 9=12015551212

To use a system name, enter:

cu systemname

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Devices
/usr/spool/locks/LCK..(tty-device)

SEE ALSO

cat(1), ct(1C), echo(1), stty(1), uucp(1C), uname(1).

DIAGNOSTICS

Exit code is zero for normal exit, otherwise, one.

WARNINGS

The *cu* command does not do any integrity checking on data it transfers. Data fields with special *cu* characters may not be transmitted properly. Depending on the interconnection hardware, it may be necessary to use a **~.** to terminate the conversion even if **stty 0** has been used. Non-printing characters are not dependably transmitted using either the **~%put** or **~%take** commands. *cu* between some modems will not return a login prompt immediately upon connection. A carriage return will return the prompt.

BUGS

There is an artificial slowing of transmission by *cu* during the **~%put** operation so that loss of data is unlikely.

NAME

custom − install specific portions of a XENIX package

SYNOPSIS

custom [**−s** set] [**−irl** packages] [**−f** file] [**−d** device]

DESCRIPTION

With *custom* you can create a custom installation by selectively installing or deleting portions of XENIX packages to or from the INTERACTIVE UNIX Operating System. The *custom* command is executable only by **root** (the superuser). It can be used interactively or it can be invoked from the command line with applicable command options.

Files are extracted or deleted in *packages*. A package is a collection of individual files. Packages are grouped in *sets*.

When in interactive mode, *custom* prompts you for the density of the diskettes of the new product distribution and for which disk drive to use if there is more than one drive. It then prompts for volume 1 and extracts the product information necessary to support it. The following menu provides support for adding or removing a package:

1. Install one or more packages
2. Remove one or more packages
3. List the available packages
4. List the files in a package
5. Install a single file
6. Select a new set to customize
7. Display current disk usage
8. Help

The following describes what action occurs with the selection of a menu option:

1. Install a package

    Prompts for one or more package names.

    Prompts for the density of the diskettes and which disk drive to use if there is more than one drive (unless that information has already been provided for this set).

    Calculates which installation volumes (distribution media) are needed, then prompts for the correct volume numbers. If multiple packages are specified, the names should be separated by spaces on the command line.

    This option, as well as "2" and "3," displays a list of all available packages in the currently selected set. Each line describes the package name; whether the package is fully installed, not installed, or partially installed; the size of the package (in 512 byte blocks); and a one-line description of the package contents.

2. Remove a package

    Prompts for one or more package names.

    Deletes the correct files in the specified package. If multiple packages are specified, the names should be separated by spaces on the command line.

    Display available packages (see option "1").

3. List available packages

    Lists the available packages in the specified set.

4. List files in a package

    Lists all files in the specified package.

    Prompts for one or more package names. Enter the name of the desired package(s).

    Displays available packages (see option "1").

5. Install a single file

    Extracts the specified file from the distribution set.

    File name should be a full path name relative to the **root** directory (/).

    Prompts for the density of the diskettes and which disk drive to use if there is more than one drive (unless that information has already been provided for this set).

6. Select a new set

    Allows you to work from a different set than the current one.

7. Display current disk usage

    Tells you your current disk usage.

8. Help

    Displays instructions to help you use *custom*.

To use *custom* from the command line, you must include the necessary information using one of the following options:

−s A set identifier
−i Install the specified package(s)
−r Remove the specified package(s)
−l List the files in the specified package(s)
−f Install the specified file
−d The full device name of the disk drive to use (e.g., /dev/rfd096ds15)

If any information is missing from the command line, *custom* prompts for the missing data.

FILES

/etc/perms/*

SEE ALSO

df(1M), du(1M), fixperm(1M), installpkg(1).

NOTES

If you insert an invalid product or a volume out of order, you will be prompted to reinsert the correct volume.

Packages containing device drivers cannot be installed using *custom*.

The *custom* command has been provided for use with any existing XENIX packages you may have that you wish to install on the INTERACTIVE UNIX Operating System. To install UNIX System packages, use *installpkg*(1).

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

cut – cut out selected fields of each line of a file

SYNOPSIS

cut −clist [file ...]
cut −flist [**−d**char] [**−s**] [file ...]

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. In addition, a file name of "−" explicitly refers to standard input.

The meanings of the options are:

list A comma-separated list of integer field numbers (in increasing order), with optional − to indicate 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.

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

ERROR: line too long
A line can have no more than 1023 characters or fields, or there is no new-line character.

ERROR: 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.

ERROR: no fields The *list* is empty.

ERROR: no delimiter
Missing *char* on **−d** option.

ERROR: cannot handle multiple adjacent backspaces
Adjacent backspaces cannot be processed correctly.

WARNING: cannot open <filename>
Either *filename* cannot be read or does not exist. If multiple file names are present, processing continues.

SEE ALSO

grep(1), paste(1).

NAME

date − print and set the date

SYNOPSIS

date [+ format]
date [mmddhhmm[[yy] | [ccyy]]]

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 (only by the super-user). 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; *cc* is the century minus one and is optional; *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 saving time. Only the super-user may change the date.

If the argument begins with +, the output of *date* is under the control of the user. 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. If the argument contains embedded blanks, it must be quoted (see the **EXAMPLE** section).

Specifications of native language translations of month and weekday names are supported. The language used depends on the value of the environment variable **LANGUAGE** [see *environ*(5)]. The month and weekday names used for a language are taken from strings in the file for that language in the **/lib/cftime** directory [see *cftime*(4)].

After successfully setting the date and time, *date* will display the new date according to the format defined in the environment variable **CFTIME** [see *environ*(5)].

Field Descriptors (must be preceded by a %):

- **a** abbreviated weekday name
- **A** full weekday name
- **b** abbreviated month name
- **B** full month name
- **d** day of month − 01 to 31
- **D** date as mm/dd/yy
- **e** day of month − 1 to 31 (single digits are preceded by a blank)
- **h** abbreviated month name (alias for %b)
- **H** hour − 00 to 23
- **I** hour − 01 to 12
- **j** day of year − 001 to 366
- **m** month of year − 01 to 12
- **M** minute − 00 to 59

- **n** insert a new-line character
- **p** string containing ante-meridiem or post-meridiem indicator (by default, AM or PM)
- **r** time as *hh:mm:ss pp* where *pp* is the ante-meridiem or post-meridiem indicator (by default, AM or PM)
- **R** time as hh:mm
- **S** second – 00 to 59
- **t** insert a tab character
- **T** time as *hh:mm:ss*
- **U** week number of year (Sunday as the first day of the week) – 01 to 52
- **w** day of week – Sunday = 0
- **W** week number of year (Monday as the first day of the week) – 01 to 52
- **x** Country-specific date format
- **X** Country-specific time format
- **y** year within century – 00 to 99
- **Y** year as *ccyy* (4 digits)
- **Z** timezone name

EXAMPLE

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

would have generated as output:

```
DATE: 08/01/76
TIME: 14:45:05
```

DIAGNOSTICS

No permission	if you are not the super-user and you try to change the date
bad conversion	if the date set is syntactically incorrect
bad format character	if the field descriptor is not recognizable.

FILES

/dev/kmem

NOTE

Administrators should note the following: if you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight time is starting or ending), and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

SEE ALSO

cftime(4), environ(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

dc — desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

The *dc* command 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.

s*x*
: 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.

l*x*
: 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.

Q
: Exits the program. 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.

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*(1) for array operations.

EXAMPLE

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
0sa1
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 [**−s**X] [**−a**n] [**−d**] [**−v**] [**−f**fsize[:isize]] inputfs outputfs

DESCRIPTION

The *dcopy* command copies file system *inputfs* to *outputfs*. *Inputfs* is the device file for the existing file system; *outputfs* is the device file to hold the reorganized result. For the most effective optimization, *inputfs* should be the raw device and *outputfs* should be the block device. Both *inputfs* and *outputfs* should be unmounted file systems.

With no options, *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:

−s*X* 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).

−a*n* 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.

−f*fsize*[:*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, followed by an interrupt or quit.

SEE ALSO

fsck(1M), mkfs(1M), ps(1).

NAME

dd − convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

The *dd* command 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.

option	*values*
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input blocks before starting copy
seek=*n*	seek *n* blocks from beginning of output file before copying
count=*n*	copy only *n* input blocks
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input block to *ibs*
...,...	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 multiplication.

The **cbs** is used only if **conv=ascii** or **conv=ebcdic** is specified. In the former case, **cbs** characters are placed into the conversion buffer (converted to ASCII). Trailing blanks are trimmed and a 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). Blanks are added to make up an output block of size **cbs**.

After completion, *dd* reports the number of whole and partial input and output blocks.

DIAGNOSTICS

f+p blocks in(out)	numbers of full and partial blocks read (written)

NAME

deluser – remove a login from the system

SYNOPSIS

deluser loginid Yes/No logdir

DESCRIPTION

deluser is used to remove a user id from the system. Once the id has been removed, the user will no longer have access to the system.

loginid This is the login that is being removed from the system.

Yes | *No* This argument determines whether or not the user's files should be saved when the login is removed.

If *Yes* is specified, all files in the user's home directory will be copied to **/lost+found/<loginid>** before the home directory is removed.

If *No* is specified, the files in $HOME will not be removed.

logdir This is the user's home directory.

NAME

deroff − remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [−mx] [−w] [files]

DESCRIPTION

The *deroff* command reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between **.EQ** and **.EN** lines and between delimiters), and *tbl*(1) descriptions, perhaps replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output.

Note: *troff*(1), *nroff*(1), and *eqn*(1) are not part of this UNIX system release.

deroff follows chains of included files (**.so** and **.nx** *troff* commands); if a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

The **−m** option may be followed by an **m**, **s**, or **l**. The **−mm** option causes the macros to be interpreted so that only running text is output (i.e., no text from macro lines). The **−ml** option forces the **−mm** option and also causes deletion of lists associated with the **mm** macros.

If the **−w** option is given, the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original with the deletions mentioned above. In text, a "word" is any string that *contains* at least two letters and is composed of letters, digits, ampersands (**&**), and apostrophes ('); in a macro call, however, a "word" is a string that *begins* with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from "words."

BUGS

deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.
The **−ml** option does not handle nested lists correctly.

NAME

devnm − device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

The *devnm* command identifies the special file associated with the mounted file system where the argument *name* resides.

This command is most commonly used by **/etc/brc** [see *brc*(1M)] to construct a mount table entry for the **root** device.

EXAMPLE

The command:

    /etc/devnm /usr

produces

    /dev/dsk/0s3

if **/usr** is mounted on **/dev/dsk/0s3**.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

brc(1M).

NAME

df — report number of free disk blocks and inodes

SYNOPSIS

df [-lt] [-f] [*file-system* | *directory* | *mounted-resource*]

DESCRIPTION

The *df* command prints out the number of free blocks and free inodes in mounted file systems, directories, or mounted resources by examining the counts kept in the super-blocks.

The *file-system* may be specified either by device name (e.g., **/dev/dsk/0s1**) or by mount point directory name (e.g., /**usr**).

directory can be a directory name. The report presents information for the device that contains the directory.

mounted-resource can be a remote resource name. The report presents information for the remote device that contains the resource.

If no arguments are used, the free space on all locally and remotely mounted file systems is printed.

The *df* command uses the following options:

−l only reports on local file systems.

−t causes the figures for total allocated blocks and inodes to be reported as well as the free blocks and inodes.

−f an actual count of the blocks in the free list is made, rather than taking the figure from the super-block (free inodes are not reported). This option will not print any information about mounted remote resources.

−v reports percent of blocks used as well as the number of blocks used and free.

NOTE

If multiple remote resources are listed that reside on the same file system on a remote machine, each listing after the first one will be marked with an asterisk.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

mount(1M).
fs(4), mnttab(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

diff − differential file comparator

SYNOPSIS

diff [−efbh] file1 file2

DESCRIPTION

The *diff* command 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:

n1 a n3,n4
n1,n2 d n3
n1,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 *n1* = *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 **−h**

SEE ALSO

bdiff(1), 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

The *diff3* command 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* = *n*2, 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.

NAME

dircmp − directory comparison

SYNOPSIS

dircmp [−d] [−s] [−w n] dir1 dir2

DESCRIPTION

The *dircmp* command 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.

−w*n* Change the width of the output line to *n* characters. The default width is 72.

SEE ALSO

cmp(1), diff(1).

NAME

diskconf – get or set disk driver parameters

SYNOPSIS

diskconf [**–g**] [**–s** code] dev

DESCRIPTION

diskconf reports on the configuration parameters of the device *dev*. When no options are used, these parameters are reported in a human readable format. However, when **–g** is specified, a coded string is produced instead. This string may then be used in subsequent invocations as the argument to the **–s** option, where it is used to update the disk driver's parameters. No sanity checking is performed in this case, although the disk driver itself may do some value checking.

The coded string is formatted as:

heads:cyls:sectors:secsize:secovhd:rsrvdcyls:intlv:skew:pheads:pcyls:psecs:flags

The definitions of the configuration parameters are:

heads — The number of tracks per cylinder.

cyls — The total number of cylinders.

sectors — The number of sectors per track.

secsize — The number of bytes per sector of user data.

secovhd — The number of bytes used for sector headers. These are normally unavailable to the user.

rsrvdcyls — The number of cylinders reserved by the disk controller for its own purposes. Normally these cylinders are not a part of the disk that is available for user data.

intlv — The interleave factor with which the disk was formatted. If unknown by the driver, 0.

skew — The skew factor between successive tracks within a cylinder with which the disk was formatted. If the interleave is not 1, this value is irrelevant; if its true value is unknown by the driver, 0 is reported.

pheads — The physical number of heads on the device; this is different from heads if the device operates with a virtual geometry. See *disk*(7) for more information.

pcyls — The physical number of cylinders in the disk.

psecs — The physical number of sectors in a track.

flags — A 4-digit hexadecimal number followed by at most 4 characters: **C, H, T,** or **S.** These indicate that the driver can be programmed to change its notion of the number of cylinders, heads, sectors per track, and sector size, respectively.

DIAGNOSTICS

A nonzero return code is the *errno* value returned from the system call that caused the failure, and an appropriate message is written to *stderr*. Usually failure results from an invalid or nonexistant *dev* or from the **V_CONFIG** *ioctl* call that failed.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

diskscan – scan a disk for bad blocks

SYNOPSIS

diskscan [–W] dev

DESCRIPTION

diskscan performs a verification pass over a formatted device specified by *dev.* The **–W** option specifies a more robust and *destructive* scan.

DIAGNOSTICS

The program maintains a simple display of progress on *stdout*. Detected bad sector numbers are written, new-line character separated, to *stderr.*

BUGS

diskscan performs only the simplest of scans, even when **–W** is specified. It is likely to find only the grossest of disk errors. A truly thorough scan could take days, and still might not find defects that appear at extremes of temperature, humidity, and electrical supply. It is best if the system administrator adheres to the manufacturer's defect list and performs frequent backups.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

diskusg – generate disk accounting data by user ID

SYNOPSIS

diskusg [options] [files]

DESCRIPTION

diskusg generates intermediate disk accounting information from data in *files*, or the standard input if omitted. *diskusg* outputs lines on the standard output, one per user, in the following format: uid login #blocks

where

uid	the numerical user ID of the user.
login	the login name of the user; and
#blocks	the total number of disk blocks allocated to this user.

diskusg normally reads only the inodes of file systems for disk accounting. In this case, *files* are the special filenames of these devices.

diskusg recognizes the following options:

–s	the input data is already in *diskusg* output format. *diskusg* combines all lines for a single user into a single line.
–v	verbose. Print a list on standard error of all files that are charged to no one.
–i *fnmlist*	ignore the data on those file systems whose file system name is in *fnmlist*. *Fnmlist* is a list of file system names separated by commas or enclosed within quotes. *diskusg* compares each name in this list with the file system name stored in the volume ID [see *labelit*(1M)].
–p *file*	use *file* as the name of the password file to generate login names. **/etc/passwd** is used by default.
–u *file*	write records to *file* of files that are charged to no one. Records consist of the special file name, the inode number, and the user ID.

The output of *diskusg* is normally the input to *acctdisk* [see *acct*(1M)] which generates total accounting records that can be merged with other accounting records. *diskusg* is normally run in *dodisk* [see *acctsh*(1M)].

EXAMPLES

The following will generate daily disk accounting information:

```
for i in /dev/dsk/0s1 /dev/dsk/0s3; do
        diskusg $i > dtmp.`basename $i` &
done
wait
diskusg -s dtmp.* | sort +0n +1 | acctdisk > disktacct
```

FILES

/etc/passwd used for user ID to login name conversions

SEE ALSO

acct(1M), acctsh(1M).

acct(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

displaypkg — display installed packages

SYNOPSIS

displaypkg

DESCRIPTION

The *displaypkg* command will list the names of all the packages that were installed using the *installpkg* command.

SEE ALSO

installpkg(1), removepkg(1).

NAME

dossette – DOS file and media utility

SYNOPSIS

dossette
dosdir [dosfile]
dostype dosfile
doscopy dosfile1 dosfile2
dosrename dosfile1 dosfile2
doserase dosfile [...]
dosmkdir dosdirectory [...]
dosrmdir dosdirectory [...]
dosformat [**−8** | **−9** | **−15** | **−18**] [**−1** | **−2**] [**−40** | **−80**]
    [**−320** | **−360** | **−720** | **−1.2** | **−1.44**]
    [**−b=**bootblock] [**−v=**volname] [**devspec**]
dosget [**−a**] [**−b**] dosfile [...] unixtarget
dosput [**−a**] [**−b**] unixfile [...] dostarget

DESCRIPTION

The *dossette* utility provides the ability to manipulate MS-DOS (DOS) format file systems under the UNIX System, and to move files between DOS and UNIX System file systems. It works with DOS diskettes and fixed disk partitions.

Device Specifiers

The *dossette* utility recognizes the following DOS-like device specifiers:

Drive	Device
A	**/dev/rdsk/f0d9dt** or **/dev/rdsk/f0q15dt** or **/dev/rdsk/f0q9dt** or **/dev/rdsk/f0d8dt** or **/dev/rdsk/f0q18dt** (will auto-select) (first diskette)
B	**/dev/rdsk/f1d9dt** or **/dev/rdsk/f1q15dt** or **/dev/rdsk/f1q9dt** or **/dev/rdsk/f1d8dt** or **/dev/rdsk/f1q18dt**
C	DOS partition on first fixed disk
D	DOS partition on second fixed disk
.	
.	
.	

The fixed disk contains a maximum of 15 usable partitions, numbered 2 through 16. The DOS device specifiers **C** through **Q** identify these partitions to *dossette*. The **C** device specifier is used to access partition 2. The **D** device specifier is used to access partition 3, etc.

If an attempt is made to access a diskette or fixed disk partition that is not in DOS format, *dossette* issues an error message identifying that diskette or fixed disk partition.

The current device is always set to **A** when the program begins execution.

File Names

A DOS-format file name consists of an optional device specifier (from the set **a:** through **q:**), an optional path name, and the name of the file, which can also be omitted in some cases. If the device specifier is omitted, the current device is assumed. If there is no path name, the current directory on the specified device is used. Thus, a file name of *foo* means file *foo* in the current directory on the current DOS device.

A file name of **b:/util/***foo* refers to a file named *foo* in the directory **/util** in a DOS file system on the second diskette drive. If a file name consists solely of a device specifier, which is legal only as the target of a **put** or **copy** command (see below), it refers to the current working directory on that device.

Note that the treatment of wildcard characters here follows the conventions of the UNIX System rather than those of DOS. For example, in *dossette* the single command:

    A> erase *

removes all files (except directories) in the current directory. Under DOS:

    A> erase *.*

is required to achieve the same effect.

The components of both UNIX System and DOS path names are separated by a slash (/). (The backslash character is not valid in *dossette*.) As a result, the two types of path names frequently look identical. However, commands that require both UNIX System and DOS file names in the same line have unambiguous syntax.

Interactive Commands

The *dossette* utility has both an interactive mode and a batch mode. When invoked with the name *dossette*, it comes up in interactive mode and reads DOS-like command lines from its standard input, executing each line as it is read. The program issues a prompt at the beginning of each line (e.g., **A>**, **B>**, etc.) to indicate the current device, as in DOS. The current device can be changed in familiar fashion by entering a different device letter followed immediately by a colon and new-line character.

dossette supports the following DOS-like set of commands in interactive mode:

dir [*dosfile*]
: List DOS directory information. The optional *dosfile* may be either a file or a directory. If *dosfile* is absent, the contents of the current directory on the current device are listed. If *dosfile* contains wildcard characters, directory data for all matching files is displayed.

type *dosfile*
: The DOS ASCII file *dosfile* is written to the standard output. Carriage-return/line-feed sequences are converted to new-line characters.

copy *dosfile1 dosfile2*
copy *dosfile* [...] *dosdirectory*
: Copy a file or files from DOS to DOS. If more than one source file is named, the target must be a DOS directory. Source files and targets may be on different devices or in different directories, but they must all be in DOS-format file systems.

rename *dosfile1 dosfile2*

Change the name of a DOS file. The existing file *dosfile1* is renamed to *dosfile2* in the same directory. Unlike its UNIX System counterpart *mv* (see *cp*(1)), the *dossette* **rename** command cannot move a file to a different directory or device.

erase *dosfile* [...]
del *dosfile* [...]

Remove DOS files. The named files are deleted from the DOS file system.

mkdir *dosdirectory*
md *dosdirectory*

Create a DOS subdirectory. A subdirectory is created with the name *dosdirectory*.

rmdir *dosdirectory*
rd *dosdirectory*

Remove a DOS subdirectory. The empty subdirectory *dosdirectory* is removed.

format [**−b=***bootblock*] [**−v=***volname*]
[**−8** | **−9** | **−15** | **−18**] [**−1** | **−2**] [**−40** | **−80**]
[**−320** | **−360** | **−720** | **−1.2** | **−1.44**]
[*devicespec*]

The named device (or, by default, the current device) is physically formatted and then initialized with an empty DOS file system. The device must be a diskette; formatting a fixed disk partition can be a serious matter and is a service that *dossette* does not provide.

The following parameters control the formatting operation:

−b=*bootblock*
: Specify the file that contains the DOS bootstrap for the diskette. The default file is **/usr/lib/dos.boot**.

−v=*volname*
: Write a volume label of *volname* into the empty file system.

−1 Format the diskette as single-sided.

−2 Format the diskette as double-sided. This is the default.

−8 Format the diskette with 8 sectors per track. This option is only supported for 5 ¼-inch media.

−9 Format the diskette with 9 sectors per track. This is the default. This option will work with both 3 ½-inch and 5 ¼-inch double density media.

−15 Format the diskette with 15 sectors per track. This option is only supported for 5 ¼-inch high density media.

−18 Format the diskette with 18 sectors per track. This option is only supported for 3 ½-inch media.

−40 Format the diskette with 40 tracks per side. This option is the default. This option is only supported for 5 ¼-inch media.

−80 Format the diskette with 80 tracks per side.

−320 Format the diskette to 320Kb capacity, with 2 sides, 8 sectors per track, and 40 tracks per side. This is equivalent to using **−2 −8 −40**. This is the format of PC-DOS 1.x disks.

−360 Format the diskette to 360Kb capacity, with 2 sides, 9 sectors per track, and 40 tracks per side. This is equivalent to using **−2 −9 −40**. This is the default. This is the format of PC-DOS 2.x and later low density disks.

−720 Format the diskette to 720Kb capacity, with 2 sides, 9 sectors per track, and 80 tracks per side. This is equivalent to using **−2 −9 −80**. This is the format of low density 3 ½-inch disks.

−1.2 Format the diskette to 1.2MB capacity, with 2 sides, 15 sectors per track, and 80 tracks per side. This is equivalent to using **−2 −15 −80**. This is the format of the high density 5 ¼-inch disks used on the IBM PC AT.

−1.44 Format the diskette to 1.44MB capacity, with 2 sides, 18 sectors per track, and 80 tracks per side. This is equivalent to using **−2 −18 −80**. This is the format of the high density 3 ½-inch disks used in many portables.

get [**−a** | **−b**] *dosfile unixfile*
get [**−a** | **−b**] *dosfile* [...] *unixdirectory*

Copy a file or files from DOS to the UNIX System. The DOS file *dosfile* is copied into the UNIX System file *unixfile*, or, if there are multiple source files, the target must be an existing UNIX System directory. Wildcard characters are permitted in source file names only. The **−a** (ASCII) option tells *dossette* to convert carriage-return/line-feed sequences to new-line characters during the copy. The **−b** (binary) option specifies no character translation. These options have no default. If neither option is present, the **get** command examines each file and attempts to decide whether a binary or ASCII copy should be made.

put [**−a** | **−b**] *unixfile dosfile*
put [**−a** | **−b**] *unixfile* [...] *dosdirectory*

Copy a file or files from the UNIX System to DOS. The UNIX System file *unixfile* is copied into the DOS file *dosfile*, or, if there are multiple source files, the target must be an existing DOS directory. Wildcard characters are permitted in source file names only. The **−a** (ASCII) option tells *dossette* to convert new-line characters to carriage-return/line-feed sequences during the copy. The **−b** (binary) option specifies no character translation. These options have no default. If neither option is present, the **put** command examines each file and attempts to decide whether a binary or ASCII copy should be done.

cd [*dosdirectory*]
: Change or display the DOS working directory on the current device. **cd** with no arguments writes to standard output the path name of the working directory on the current device. If an argument is present, it specifies the new working directory for the current device. Note that a separate working directory is maintained for each device.

chloc *unixdirectory*
: Change the UNIX System working directory to *unixdirectory*. This is useful when **get**ting or **put**ting files to and from several different UNIX System directories.

a: Change the default DOS device to **A**, which corresponds to the first diskette drive.

b: Change the default DOS device to **B**, which corresponds to the second diskette drive.

c: ... q: Change the default DOS device to the fixed disk partition that corresponds to the device specifier used. The **C** device specifier corresponds to the second partition on the fixed disk. The **D** device specifier corresponds to the third partition on the fixed disk. This pattern continues through device specifier **Q** which corresponds to the sixteenth and last fixed disk partition.

! *command* [*arg* ...]
: Invoke a shell to execute the *command* with the *args*. This is useful for looking at UNIX System directories and files without leaving *dossette*.

help List the available *dossette* commands on the standard output.

q Exit *dossette*.

Batch Mode Commands

The batch mode interface permits *dossette* commands to be invoked noninteractively in shell scripts. Whenever the program is invoked by a name other than *dossette*, the invoking name is interpreted as a command, and the remaining command line arguments are used as arguments to that command. The syntax of each batch mode command matches that of the corresponding interactive command.

Some interactive functions, e.g., **cd**, are not meaningful in batch mode and therefore are not supported. The batch mode commands are:

dosdir
dostype
doscopy
dosrename
doserase
dosmkdir
dosrmdir
dosformat
dosget
dosput

WARNINGS

When using wildcard characters with *dossette* batch mode commands, take care to hide them from the UNIX System shell by quoting them when it is appropriate to do so. For example:

WRONG:

```
dosget *.c /t/kens/srcdir
```

RIGHT:

```
dosget "*.c" /t/kens/srcdir
```

However:

RIGHT:

```
dosput /t/kens/srcdir/*.c a:/chris/dossrc
```

RIGHT:

```
dosput "/t/kens/srcdir/*.c" a:/chris/dossrc
```

In the first example, the "*.c" is expanded by the shell with (possibly nonexistent) UNIX System file names, yielding error messages at best and wrong results at worst. In the second example, the wild card character is quoted, and is therefore passed in to *dossette* for proper expansion. In the third and fourth sample lines shown above, however, the quoting is moot; if the source file name specification is not quoted, the shell expands it correctly into a list of file names. If it is quoted, then the wildcard character is passed to *dossette*, which expands it the same way the shell would have.

Console messages such as:

```
FD0: data overrun
```

may appear while *dossette* attempts to determine the characteristics of the diskette (such as density, number of tracks, and sectors per track.) These messages may be safely ignored.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

du – summarize disk usage

SYNOPSIS

du [−sar] [names **]**

DESCRIPTION

The *du* command reports 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, the current directory is used.

The optional arguments are as follows:

−s causes only the grand total (for each of the specified *names*) to be given.

−a causes an output line to be generated for each file.

If neither **−s** or **−a** is specified, an output line is generated for each directory only.

−r will cause *du* to generate messages about directories that cannot be read, files that cannot be opened, etc., rather than being silent (the default).

A file with two or more links is only counted once.

BUGS

If the **−a** option is not used, non-directories given as arguments are not listed.

If there are links between files in different directories where the directories are on separate branches of the file system hierarchy, *du* will count the excess files more than once.

Files with holes in them will get an incorrect block count.

NAME

echo — echo arguments

SYNOPSIS

echo [-n] [arg] ...

DESCRIPTION

The *echo* command writes its arguments separated by blanks and terminated by a new-line on the standard output. The **-n** option prints a line without the new-line; same as using the **\c** escape sequence.

echo 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
\0*n*	where *n* is the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number representing that character.

The *echo* command is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

sh(1).

CAVEATS

When representing an 8-bit character by using the escape convention **\0***n*, the *n* must **always** be preceded by the digit zero (0).

For example, typing: **echo 'WARNING:\07'** will print the phrase **WARNING:** and sound the "bell" on your terminal. The use of single (or double) quotes (or two backslashes) is required to protect the "\" that precedes the "07".

For the octal equivalents of each character, see *ascii*(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

ed, red – text editor

SYNOPSIS

ed [–s] [–p string **] [–x] [–C]** [file]

red [–s] [–p string **] [–x] [–C]** [file]

DESCRIPTION

ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see the following text) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited.

–s 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*.

–p Allows the user to specify a prompt string.

–x Encryption option; when used, *ed* simulates an **X** command and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of *crypt*(1). The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **–x** option. See *crypt*(1). Also, see the **WARNINGS** section at the end of this manual page.

–C Encryption option; the same as the **–x** option, except that *ed* simulates a **C** command. The **C** command is like the **X** command, except that all text read in is assumed to have been encrypted.

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 allow editing of files only 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: When you are entering text into the file, this format is not in effect; instead, because of being in **stty –tabs** or **stty tab3** mode, tabs are expanded to every eighth column.

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. Leave input mode by typing a period (.) at the beginning of a line, followed immediately by a carriage return.

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. $ (dollar sign), 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 *RE*s 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 dollar sign (**$**) 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 an ASCII lower-case letter (**a**-**z**). 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**.
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**.
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 *a*ppend 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 *c*hange 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.

C

Same as the **X** command, except that *ed* assumes all text read in for the *e* and *r* commands is encrypted unless a null key is typed in.

(.,.)**d**

The *d*elete 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 *e*dit 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.**

E *file*

The *E*dit 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 *f*ile-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 *g*lobal 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 multi-line 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.**

(1,$)G/*RE***/**

In the interactive *G*lobal 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 *h*elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H

The *H*elp 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 *i*nsert 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 *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be an ASCII lower-case letter (**a-z**). The address '*x* then addresses this line; **.** is unchanged.

(.,.)l

The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab, backspace*) are represented by visually mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any command other than *e*, *f*, *r*, or *w*.

(.,.)m*a*

The *m*ove 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 *n*umber 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 command other than *e*, *f*, *r*, or *w*.

(.,.)p

The *p*rint command prints the addressed lines; **.** is left at the last line printed. The *p* command may be appended to any 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 *q*uit command causes *ed* to exit. No automatic write of a file is done; however, see **DIAGNOSTICS**.

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

($)r *file*

The *r*ead 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** or
(**.**,**.**)**s**/*RE*/*replacement*/*n* $n = 1-512$

The *s*ubstitute 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. If a number *n* appears after the command, only the *n*-th occurrence of the matched string on each addressed line 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.**

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.

(**.**,**.**)**t***a*

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 *u*ndo 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 *w*rite 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 *umask*(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.

X

A key is prompted for, and it is used in subsequent *e*, *r*, and *w* commands to decrypt and encrypt text using the *crypt*(1) algorithm. An educated guess is made to determine whether text read in for the *e* and *r* commands is encrypted. A null key turns off encryption. Subsequent *e*, *r*, and *w* commands will use this key to encrypt or decrypt the text [see *crypt*(1)]. An explicitly empty key turns off encryption. Also, see the **−x** option of *ed*.

($)=

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 system 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 in a line, 256 characters in a global command list, and 64 characters in the path name of a file (counting slashes). 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.

If a file is not terminated by a new-line character, *ed* adds one and puts out a message explaining what it did.

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

$TMPDIR	if this environmental variable is not null, its value is used in place of **/usr/tmp** as the directory name for the temporary work file.
/usr/tmp	if **/usr/tmp** exists, it is used as the directory name for the temporary work file.
/tmp	if the environmental variable **TMPDIR** does not exist or is null, and if **/usr/tmp** does not exist, then **/tmp** is used as the directory name for the temporary work file.
ed.hup	work is saved here if the terminal is hung up.

NOTES

The − option, although it continues to be supported, has been replaced in the documentation by the **−s** option that follows the Command Syntax Standard [see *intro*(1)].

SEE ALSO

edit(1), ex(1), grep(1), sed(1), sh(1), stty(1), umask(1), vi(1).
fspec(4), regexp(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

DIAGNOSTICS

?	for command errors.
?file	for an inaccessible file. (use the *h*elp and *H*elp 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 **−s** command-line option inhibits this feature.

WARNINGS

The encryption options and commands are provided with the Security Administration Utilities package, which is available only in the United States.

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.

If the editor input is coming from a command file (e.g., ed file < ed-cmd-file), the editor will exit at the first failure.

NAME

edit – text editor (variant of ex for casual users)

SYNOPSIS

edit [–r] [–x] [–C] 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. It operates precisely as *ex*(1) with the following options automatically set:

novice	ON
report	ON
showmode	ON
magic	OFF

These options can be turned on or off via the *set* command in *ex*(1).

–r Recover file after an editor or system crash.

–x Encryption option; when used, the file will be encrypted as it is being written and will require an encryption key to be read. *edit* makes an educated guess to determine if a file is encrypted or not. See *crypt*(1). Also, see the **WARNING** section at the end of this manual page.

–C Encryption option; the same as **–x** except that *edit* assumes files are encrypted.

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*.

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 that you can then edit, and tells you how many lines and characters are in the file. To create a new file, you also begin with the command **edit** with a filename: **edit** *name*; the editor will tell you it is a New File.

The *edit* command prompt is the colon (:), 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). When you start editing, *edit* makes the last line of the file the current line. Most commands to *edit* use the current line if you do not tell them which line to use. Thus if you say *print* (which can be abbreviated *p*) and type carriage return (as you should after all *edit* commands), the current line will be printed. If you *delete* (*d*) the current line, *edit* will print the new current line, which is usually the next line in the file. If you *delete* the last line, then the new last line becomes the current one.

If you start with an empty file or wish to add some new lines, then the *append* (*a*) command can be used. After you execute this command (typing a carriage return after the word *append*), *edit* will read lines from your terminal until you type a line consisting of just a dot (.); it places 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 type before, rather than after, the current line.

edit numbers the lines in the buffer, with the first line having number 1. If you execute the command *1*, then *edit* will type the first line of the buffer. If you then execute the command *d*, *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: **s**/*old*/*new*/ where *old* is the string of characters you want to replace and *new* is the string of characters you want to replace *old* 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 the buffer. After modifying a file, you can save the contents of the file by executing a *write* (*w*) command. You can 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 receive the message No write since last change (:quit! overrides), and *edit* will wait for another command. If you do not want to write the buffer out, issue the *quit* command followed by an exclamation point (*q*!). The buffer is then irretrievably discarded and you return to the shell.

By using the *d* and *a* commands and giving line numbers to see lines in the file, you can make any changes you want. You should learn at least a few more things, however, if you will use *edit* more than a few times.

The *change* (*c*) command changes the current line to a sequence of lines you supply (as in *append*, you type lines up to a line consisting of only a dot (**.**). You can tell *change* to change more than one line by giving the line numbers of the lines you want to change, i.e., **3,5c**. You can print lines this way too: **1,23p** prints the first 23 lines of the file.

The *undo* (*u*) command reverses the effect of the last command you executed that changed the buffer. Thus if you execute a *substitute* command that does not do what you want, type *u* and the old contents of the line will be restored. You can also undo an *undo* command. *edit* will give you a warning message when a command affects more than one line of the buffer. Note that commands such as *write* and *quit* cannot be undone.

To look at the next line in the buffer, type carriage return. To look at a number of lines, type **^D** (while holding down the control key, press **d**) 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 nearby text by executing the *z* command. The current line will appear in the middle of the text displayed, and the last line displayed will become the current line; you can get back to the line where you were before you executed the *z* command by typing ''. The *z* command has other options: *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 *z.11* to display five lines before and five lines after the current line. (Typing **z.***n*, when *n* is an odd number, displays a total of *n* lines, centered about the current line; when *n* is an even number, it displays *n*−1 lines, so that the lines displayed are centered around the current line.) You can give counts after other commands; for example, you can delete 5 lines starting with the current line with the command **d5**.

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 *text*, it wraps 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 the symbolic name dot (**.**); this is most useful in a range of lines as in **.,$p** which prints the current line plus the rest of the lines in the file. To move to the last line in the file, you can refer to it by its symbolic name **$**. Thus the command **$d** deletes the last line in the file, no matter what the current line is. Arithmetic with line references is also possible. Thus the line **$−5** is the fifth before the last and **.+20** is 20 lines after the current line.

You can determine the current line by typing **.=**. This is useful if you wish to move or copy a section of text within a file or between files. Find the first and last line numbers you wish to copy or move. To move lines 10 through 20, type **10,20d a** to delete these lines from the file and place them in a buffer named **a**. *edit* has 26 such buffers named **a** through **z**. To put the contents of buffer **a** after the current line, type *put a*. If you want to move or copy these lines to another file, execute an *edit* (*e*) command after copying the lines; following the *e* command with the name of the other file you wish to edit, i.e., *edit chapter2*. To copy lines without deleting them, use *yank* (*y*) in place of **d**. If the text you wish to move or copy is all within one file, it is not necessary to use named buffers. For example, to move lines 10 through 20 to the end of the file, type **10,20m $**.

SEE ALSO

ed(1), ex(1), vi(1).

WARNING

The encryption options are provided with the Security Administration Utilities package, which is available only in the United States.

NAME

egrep – search a file for a pattern using full regular expressions

SYNOPSIS

egrep [options] full regular expression [file ...]

DESCRIPTION

The *egrep* command (*expression grep*) searches files for a pattern of characters and prints all lines that contain that pattern. *egrep* uses full regular expressions (expressions that have string values that use the full set of alphanumeric and special characters) to match the patterns. It uses a fast deterministic algorithm that sometimes needs exponential space.

The *egrep* command accepts full regular expressions as in *ed*(1), except for \(and \), with the addition of:

1. A full regular expression followed by + that matches one or more occurrences of the full regular expression.
2. A full regular expression followed by ? that matches 0 or 1 occurrences of the full regular expression.
3. Full regular expressions separated by | or by a new-line that match strings that are matched by any of the expressions.
4. A full regular expression that may be enclosed in parentheses () for grouping.

Be careful using the characters $, *, [, ^, |, (,), and \ in *full regular expression*, because they are also meaningful to the shell. It is safest to enclose the entire *full regular expression* in single quotes '...'.

The order of precedence of operators is [ ], then * ? +, then concatenation, then | and new-line.

If no files are specified, *egrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

–b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

–c Print only a count of the lines that contain the pattern.

–i Ignore upper/lower case distinction during comparisons.

–l Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.

–n Precede each line by its line number in the file (first line is 1).

–v Print all lines except those that contain the pattern.

–e *special_expression*
Search for a *special expression* (*full regular expression* that begins with a –).

–f *file*
Take the list of *full regular expressions* from *file*.

-h Prevents the name of the file containing the matching lines from being appended to that line. Used when searching multiple files.

SEE ALSO

ed(1), fgrep(1), grep(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* command, but there is not 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**, which is included as part of the basic software development set.

NAME

enable, disable – enable/disable LP printers

SYNOPSIS

enable *printers*
disable [*options*] *printers*

DESCRIPTION

The *enable* command activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

The *disable* command 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 for use with *disable* are:

−c Cancel any requests that are currently printing on any of the designated printers. This option cannot be used with the **−W** option.

−r *reason* Assign a *reason* for the disabling of the printers. This *reason* applies to all printers mentioned up to the next **−r** option. This *reason* is reported by *lpstat*(1). If the **−r** option is not present, then a default reason will be used.

−W Disable the specified printers when the print requests currently printing have finished. This option cannot be used with the **−c** option.

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).

NAME

env — set environment for command execution

SYNOPSIS

env [—] [name=value] ... [command args]

DESCRIPTION

The *env* command 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 *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

ex – text editor

SYNOPSIS

ex [–s] [–v] [–t tag**] [–r** file**] [–L] [–R] [–x] [–C] [–c** command**]** file ...

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*.

For ed Users

If you have used *ed*(1), you will find that, in addition to having all of the *ed*(1) commands available, *ex* has a number of additional features useful on CRT terminals. Intelligent terminals and high-speed terminals are very pleasant to use with *vi*. Generally, the *ex* editor uses far more of the capabilities of terminals than *ed*(1) does and uses the terminal capability data base [see *terminfo*(4)] and the type of the terminal you are using from the environmental variable TERM 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 features for easily viewing the text of the file. The *z* command gives easy access to windows of text. Typing **^D** (control-d) causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just typing return. Of course, the screen-oriented *visual* mode gives constant access to editing context.

ex gives you 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 overwrite a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the telephone, you can use the editor *recover* command (or **–r** *file* option) 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.

The editor has a group of buffers whose names are the ASCII lower-case letters (**a-z**). You can place text in these named buffers where it is available to be inserted elsewhere in the file. The contents of these buffers remain available when you begin editing a new file using the *edit* (*e*) command.

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 the 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 that allows the editor to supply leading white space to align text automatically. You can then use **^D** as a backtab and space or tab to move forward to align new code easily.

Miscellaneous useful features include an intelligent *join* (*j*) command that 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*(1).

Invocation Options

The following invocation options are interpreted by *ex* (previously documented options are discussed in the **NOTES** section at the end of this manual page):

−s Suppress all interactive-user feedback. This is useful in processing editor scripts.

−v Invoke *vi*

−t *tag* Edit the file containing the *tag* and position the editor at its definition.

−r *file* Edit *file* after an editor or system crash. (Recovers the version of *file* that was in the buffer when the crash occurred.)

−L List the names of all files saved as the result of an editor or system crash.

−R **Readonly** mode; the **readonly** flag is set, preventing accidental overwriting of the file.

−x Encryption option; when used, *ex* simulates an **X** command and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of *crypt*(1). The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **−x** option. [See *crypt*(1)]. Also, see the **WARNINGS** section at the end of this manual page.

−C Encryption option; the same as the **−x** option, except that *ex* simulates a **C** command. The **C** command is like the **X** command, except that all text read in is assumed to have been encrypted.

−c *command* Begin editing by executing the specified editor *command* (usually a search or positioning command).

The *file* argument indicates one or more files to be edited.

ex States

Command	Normal and initial state. Input prompted for by **:**. Your line kill character cancels a partial command.
Insert	Entered by **a**, **i**, or **c**. Arbitrary text may be entered. Insert state normally is terminated by a line having only "**.**" on it, or, abnormally, with an interrupt.
Visual	Entered by typing **vi**; terminated by typing **Q** or **^** (control-\\).

ex Command Names and Abbreviations

abbrev	**ab**	**map**		**set**	**se**
append	**a**	**mark**	**ma**	**shell**	**sh**
args	**ar**	**move**	**m**	**source**	**so**
change	**c**	**next**	**n**	**substitute**	**s**
copy	**co**	**number**	**nu**	**unabbrev**	**unab**
delete	**d**	**preserve**	**pre**	**undo**	**u**
edit	**e**	**print**	**p**	**unmap**	**unm**
file	**f**	**put**	**pu**	**version**	**ve**
global	**g**	**quit**	**q**	**visual**	**vi**
insert	**i**	**read**	**r**	**write**	**w**
join	**j**	**recover**	**rec**	**xit**	**x**
list	**l**	**rewind**	**rew**	**yank**	**ya**

ex Commands

forced encryption	**C**	heuristic encryption	**X**
resubst	**&**	print next	CR
rshift	**>**	lshift	**<**
scroll	**^D**	window	**z**
shell escape	**!**		

ex Command Addresses

n	line *n*	*/pat*	next with *pat*
.	current	*?pat*	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
−	previous	*'x*	marked with *x*
+*n*	*n* forward	*''*	previous context
%	1,$		

Initializing options

EXINIT	place **set**'s here in environment variable
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set *x*	enable option *x*

set no*x*	disable option *x*
set *x***=***val*	give value *val* to option *x*
set	show changed options
set all	show all options
set *x***?**	show value of option *x*

Most useful options and their abbreviations

autoindent	**ai**	supply indent
autowrite	**aw**	write before changing files
directory	**dir**	specify the directory
exrc	**ex**	allow *vi/ex* to read the **.exrc** in the current directory. This option is set in the **EXINIT** shell variable or in the **.exrc** file in the **$HOME** directory.
ignorecase	**ic**	ignore case of letters in scanning
list		print **^I** for tab, **$** at end
magic		treat **. [*** special in patterns
modelines		first five lines and last five lines executed as *vi/ex* commands if they are of the form **ex:***command***:** or **vi:***command***:**
number	**nu**	number lines
paragraphs	**para**	macro names that start paragraphs
redraw		simulate smart terminal
report		informs you if the number of lines modified by the last command is greater than the value of the **report** variable
scroll		command mode lines
sections	**sect**	macro names that start sections
shiftwidth	**sw**	for **< >**, and input **^D**
showmatch	**sm**	to) and } as typed
showmode	**smd**	show insert mode in *vi*
slowopen	**slow**	stop updates during insert
term		specifies to **vi** the type of terminal being used (the default is the value of the environmental variable **TERM**)
window		visual mode lines
wrapmargin	**wm**	automatic line splitting
wrapscan	**ws**	search around end (or beginning) of buffer

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[*str***]**	any character in *str*
[^*str***]**	any character not in *str*
[*x−y***]**	any character between *x* and *y*
*****	any number of preceding characters

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/exstrings	error messages
/usr/lib/exrecover	recover command
/usr/lib/expreserve	preserve command
/usr/lib/terminfo/*	describes capabilities of terminals
$HOME/.exrc	editor startup file
./.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve/*login*	preservation directory (where *login* is the user's login)

NOTES

Several options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard [see *intro*(1)]. The − option has been replaced by **−s**, a **−r** option that is not followed with an option-argument has been replaced by **−L**, and *+command* has been replaced by **−c** *command*.

SEE ALSO

crypt(1), ed(1), edit(1), grep(1), sed(1), sort(1), vi(1).

curses(3X), term(4), terminfo(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

User's Guide.

"curses/terminfo" chapter of the *Programmer's Guide*.

WARNINGS

The encryption options and commands are provided with the Security Administration Utilities package, which is available only in the United States.

BUGS

The *z* command prints the number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line **−s** 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.

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).

DIAGNOSTICS

As 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.

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

factor — obtain the prime factors of a number

SYNOPSIS

factor [integer]

DESCRIPTION

When you use *factor* without an argument, it waits for you to give it an integer. After you give it a positive integer less than or equal to 10^{14}, it factors the integer, prints its prime factors the proper number of times, and then waits for another integer. *factor* exits if it encounters a zero or any non-numeric character.

If you invoke *factor* with an argument, it factors the integer as described above, and then it exits.

The maximum time to factor an integer is proportional to $\sqrt{n}$. *factor* will take this time when *n* is prime or the square of a prime.

DIAGNOSTICS

factor prints the error message, "Ouch," for input out of range or for garbage input.

NAME

fdisk – create or modify fixed disk partition table

SYNOPSIS

fdisk [argument]

DESCRIPTION

This command is used to create and modify the partition table that is put in the first sector of the fixed disk. This table is used by MS-DOS (DOS) and by the first-stage bootstrap to identify parts of the disk reserved for different operating systems, and to identify the partition containing the second-stage bootstrap (the *active* partition). The optional argument can be used to specify the raw device associated with the fixed disk; the default value is **/dev/rdsk/0s0**.

The program displays the partition table as it exists on the disk, and then presents a menu allowing the user to modify the table. The menu, questions, warnings, and error messages are intended to be self-explanatory.

If there is no partition table on the disk, the user is given the option of creating a default partitioning or specifying the initial table values. The default partitioning allocates 10 percent of the disk for DOS and 90 percent for the UNIX System, and makes the UNIX System partition active. In either case, when the initial table is created, *fdisk* also writes out the first-stage bootstrap code (see *disk*(7)) along with the partition table. After the initial table is created, only the table is changed; the bootstrap is not modified.

Upon start-up, the *fdisk* command will check any extended DOS partition entries, and if they need correcting before they can be used by the UNIX System, you will be prompted:

```
Do you want to fix the Extended DOS Partitions for UNIX access?
Please type "y" or "n":
```

Type **y** to correct the information for the extended DOS partitions. Type **n** if you do not want to correct the information.

WARNING: If you do not correct the information but you still try to mount an extended DOS partition, the system will mount the *primary* DOS partition as the extended DOS partition, thereby causing great damage to your system. To make this change take effect, you must reboot the system.

The DOS partition types are currently listed as DOS12 and DOS16 for the type of file allocation table (FAT) used on each DOS partition.

Menu Options

The menu options given by the *fdisk* program are:

Create a partition

This option allows the user to create a new partition. The maximum number of partitions is 4. The program will ask for the type of the partition (MS-DOS, UNIX System, or other). It will then ask for the size of the partition as a percentage of the disk. The user may also enter the letter **c** at this point, in which case the program will ask for the starting cylinder number and size of the partition in cylinders. If a **c** is not entered, the program will determine the starting cylinder number where the partition will fit. In either case, if the partition would overlap an existing partition or will not fit, a message is displayed and the program returns to the original menu.

Change Active (Boot from) partition
: This option allows the user to specify the partition where the first-stage bootstrap will look for the second-stage bootstrap, otherwise known as the *active* partition.

Delete a partition
: This option allows the user to delete a previously created partition. Note that this will destroy all data in that partition.

Display partition table
: This option allows the user to display the primary partition list. If there are any extended logical DOS partitions, the system will prompt:

```
Do you want to display the Extended DOS Partitions?
Please type "y" or "n":
```

Type **y** to display the extended DOS partition. Type **n** to return to the main menu. If you type **y**, the system will then prompt:

```
Hit return to continue
```

The system will display a partition number. This partition number is the name that the DOS partition must be mounted with, i.e., **/dev/dsk/0p5** to mount the first extended DOS partition. Use the following options to include your modifications to the partition table at this time or to cancel the session without modifying the table:

Exit
: This option writes the new version of the table created during this session with *fdisk* out to the fixed disk, and exits the program.

Cancel
: This option exits without modifying the partition table.

DIAGNOSTICS

Most messages will be self-explanatory. The following may appear immediately after starting the program:

Fdisk: *cannot open <device>*
: This indicates that the device name argument is not valid.

Fdisk: *unable to get device parameters for device <device>*
: This indicates a problem with the configuration of the fixed disk, or an error in the fixed disk driver.

Fdisk: *error reading partition table*
: This indicates that some error occurred when trying initially to read the fixed disk. This could be a problem with the fixed disk controller or driver, or with the configuration of the fixed disk.

Fdisk: *error fixing disk*
: This indicates that an error occurred when trying to correct the extended DOS partition information.

Fdisk: *error reading extended DOS partition table*
This indicates that some error occurred when trying initially to read the fixed disk. This could be a problem with the fixed disk controller or driver, or with the configuration of the fixed disk.

These messages may appear after selecting the *Exit* option from the menu:

Fdisk: *error writing boot record*
This indicates that some error occurred when trying to write the new partition table out to the fixed disk. This could be a problem with the fixed disk controller, the disk itself, the driver, or the configuration of the fixed disk.

Fdisk: *error writing extended DOS partition table*
This indicates that some error occurred when trying initially to write to the fixed disk. This could be a problem with the fixed disk controller or driver, or with the configuration of the fixed disk.

FILES
/dev/rdsk/0s0

SEE ALSO
mkpart(1M), disk(7).

WARNING
Compatible with MS-DOS Versions 3.3 or earlier.

ADDED VALUE
This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

ff — list file names and statistics for a file system

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

The *ff* command reads the i-list and directories of the *special* file, assuming it is a file system. Inode data is saved for files which match the selection criteria. Output consists of the path name for each saved inode, plus other file information requested using the print *options* below. Output fields are positional. The output is produced in inode 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 inode number after each path name.

−l Generate a supplementary list of all path names for multiple-linked files.

−p *prefix* The specified *prefix* will be added to each generated path name. The default is **.** (dot).

−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 inode has been accessed in *n* days.

−m *n* Select if the inode has been modified in *n* days.

−c *n* Select if the inode has been changed in *n* days.

−n *file* Select if the inode has been modified more recently than the argument *file*.

−i *inode-list* Generate names for only those inodes specified in *inode-list*.

SEE ALSO

find(1), ncheck(1M).

BUGS

If the **−l** option is not specified, only a single path name out of all possible ones is generated for a multiple-linked inode. If **−l** is specified, all possible names for every linked file on the file system are included in the output. However, no selection criteria apply to the names generated.

NAME

fgrep – search a file for a character string

SYNOPSIS

fgrep [options] string [file ...]

DESCRIPTION

The *fgrep* (fast *grep*) command searches files for a character string and prints all lines that contain that string. *fgrep* is different from *grep(1)* and *egrep(1)* because it searches for a string, instead of searching for a pattern that matches an expression. It uses a fast and compact algorithm.

The characters **$**, *****, **[**, **^**, **|**, (,), and \ are interpreted literally by *fgrep,* that is, *fgrep* does not recognize full regular expressions as does *egrep*. Since these characters have special meaning to the shell, it is safest to enclose the entire *string* in single quotes '...'.

If no files are specified, *fgrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

–b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

–c Print only a count of the lines that contain the pattern.

–i Ignore upper/lower case distinction during comparisons.

–l Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.

–n Precede each line by its line number in the file (first line is 1).

–v Print all lines except those that contain the pattern.

–x Print only lines matched entirely.

–e *special_string*
Search for a *special string* (*string* begins with a –).

–f *file*
Take the list of *strings* from *file*.

–h Prevents the name of the file containing the matching line from being appended to that line. Used when searching multiple files.

SEE ALSO

ed(1), egrep(1), grep(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* command, but there is not 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**, which is included as part of the basic software development set.

NAME

file − determine file type

SYNOPSIS

file [**−c**] [**−f** ffile] [**−m** mfile] arg ...

DESCRIPTION

The *file* command 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.

−c The **−c** option 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**.

−f If the **−f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

−m The **−m** option instructs *file* to use an alternate magic file.

The *file* command 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.

FILES

/etc/magic

SEE ALSO

filehdr(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

find − find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

The *find* command recursively descends the directory hierarchy for each path name in the *path-name-list* (that is, 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*. Valid expressions are:

−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, only the bits that are set in *onum* are compared with the file permission flags, and the expression evaluates true if they match.

−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 semicolon. 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*(1) 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.

−mount Always true; restricts the search to the file system containing the directory specified, or if no directory was specified, the current directory.

−local True if the file physically resides on the local system.

(*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), umask(2), fs(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

BUGS

find / −depth always fails with the message: "find: stat failed: : No such file or directory".

NAME

fixperm – correct or initialize XENIX file permissions and ownership

SYNOPSIS

fixperm [–acDfgilnSsvw [–d*package*] **[–u** *package*]] *specfile*

DESCRIPTION

For each line in the specification file **specfile,** *fixperm* makes the listed pathname conform to a specification. *fixperm* is typically used by the super-user to configure a XENIX system upon installation. It has been provided for use with any existing XENIX packages that you may have that you wish to install on the UNIX system. Nonsuper-users can only use *fixperm* with the **–D**, **–f**, **–l**, or **–n** options.

The following options are available:

Option	**Description**
–a	All files in the perm file must exist. This means that files marked as optional (type letter is in capital letters) must be present.
–c	Creates empty files and missing directories.
–D	Lists directories only on standard output. Does not modify target files.
–d*package*	Processes input lines beginning with given package specifier string (see above). For instance, -dBASE processes only items specified as belonging to the Basic utilities set. The default action is to process all lines.
–f	Lists files only on standard output. Does not modify target files.
–g	Lists all devices on the standard output. Target files are not modified (analogous to **-l**, **-f**, and **-D**).
–i	Checks to see if the selected packages are installed. Return values are 0: package completely installed 4: package not installed 5: package partially installed If the equivalent package was installed as a UNIX package, **-i** will not detect it.
–l	Lists files and directories on standard output. Does not modify target files.
–n	Reports errors only. Does not modify target files.
–S	Issues a complaint if files are not in x.out format.
–s	Modifies special device files in addition to the rest of the permlist.
–u*package*	Causes similar action to **–d** option but processes items that are not part of the given package.

−v (verbose)	Issues a complaint if executable files are 1) word-swapped, 2) not fixed-stack, 3) not separate I and D, or 4) not stripped.
−w	Lists location (volume number) of the specified files or directories.

Specification File Format

Each nonblank line in the specification file consists of either a comment or an item specification. A comment is any text from a pound sign "#" up to the end of the line. There is one item specification per line. User and group id numbers must be specified at the top of the specification file for each user and group mentioned in the file.

An item specification consists of a package specifier, a permission specification, owner and group specifications, the number of links on the file, the filename, and an optional volume number.

The package specifier is an arbitrary string that is the name of a package within a distribution set. A package is a set of files.

A permission specification follows the package specifier. The permission specification consists of a file type, followed by a numeric permission specification. The item specification is one of the following characters:

Character	**Description**
x	executable
a	archive
e	empty file (create if **−c** option given)
b	block device
c	character device
d	directory
f	text file
p	named pipe

If the item specification is given as an uppercase letter, the file associated with it is optional, and *fixperm* will not return an error message if it does not exist.

The numeric permission conforms to the scheme described in *chmod*(1). The owner and group permissions are in the third column separated by a slash, such as "bin/bin". The fourth column indicates the number of links. If there are links to the file, the next line contains the linked filename with no other information. The fifth column is a pathname. The pathname must be relative (not preceded by a slash "/"). The sixth column is only used for special files, major and minor device numbers, or volume numbers.

EXAMPLES

The following two lines make a distribution and invoke *tar*(1) to archive only the files in **my_package** on **/dev/sample**:

```
/etc/fixperm -f /etc/perm/my_package> list
tar cfF /dev/sample list
```

This command line reports package errors:

```
/etc/fixperm -nd my_package
```

NOTES

fixperm is usually only run by a shell script at installation.

fixperm should only be run from the directory to which the target files are relative.

SEE ALSO

custom(1M).

NAME

format − format floppy disk tracks

SYNOPSIS

/bin/format [**−vVE**] [**−f** first] [**−l** last] [**−i** interleave] device[**t**]

DESCRIPTION

The *format* command formats floppy 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 track to be formatted. The default interleave of 2 may be modified by using the **−i** option. *Device* must specify a raw (character) floppy device. The **t** indicates the entire disk. Absence of this letter indicates that the first track of the diskette cannot be accessed.

−v verbose.

−V verify. After tracks are formatted, a random sector is chosen and a write of test data is done into it. The sector is then read back and a comparison is made.

−E exhaustive verify. Every sector is verified by write/read/compare.

FILES

/dev/rdsk/* raw device for partition to be formatted

SEE ALSO

mkpart(1M), fd(7).

NAME

fsck, dfsck – check and repair file systems

SYNOPSIS

/etc/fsck [**–y**] [**–n**] [**–sX**] [**–SX**] [**–t** file] [**–q**] [**–D**] [**–f**] [**–b**] [file-systems]

/etc/dfsck [options1] fsys1 ... – [options2] fsys2 ...

DESCRIPTION

fsck

The *fsck* command audits and interactively repairs inconsistent conditions for file systems. If the file system is found to be consistent, the number of files, blocks used, and blocks free are reported. If the file system is inconsistent, the user 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 loss may be determined from the diagnostic output. The default action for each correction is to wait for the user to respond **yes** or **no**. If the user does not have write permission, *fsck* defaults to a **–n** action.

The following options are accepted 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.

–s*X* Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super block 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 super block will not continue to be used, or written on the file system.

The **–s***X* option allows for creating an optimal free-list organization.

If *X* is not given, the values used when the file system was created are used. The format of *X* is *cylinder size:gap size*.

–S*X* Conditionally reconstruct the free list. This option is like **–s***X* 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 user 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.

−q Quiet *fsck*. Do not print size-check messages. Unreferenced **fifos** will silently be removed. If *fsck* requires it, counts in the super block 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 and check the free list. The free list will be reconstructed, if 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 system. 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 inode or the free list.
2. Blocks claimed by an inode 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 inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated inode.
   Inode number out of range.
8. Super Block checks:
   More than 65536 inodes.
   More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the user'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. Empty files or directories are removed, as long as the **−n** option is not specified. *fsck* will force the reconnection of nonempty directories. The name assigned is the inode 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

The *dfsck* command 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* command permits a user to interact with two *fsck* programs at once. To aid in this, *dfsck* will print the file system name for each message to the user. When answering a question from *dfsck*, the user must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

FILES

/etc/checklist contains default list of file systems to check.

SEE ALSO

crash(1M), mkfs(1M), ncheck(1M), xfsck(1M).
uadmin(2), checklist(4), fs(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

BUGS

Inode numbers for **.** and **..** in each directory are not checked for validity.

NAME

fsdb − file system debugger

SYNOPSIS

/etc/fsdb special [−]

DESCRIPTION

The *fsdb* command 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 inode. These greatly simplify the process of correcting control block entries or descending the file system tree.

The *fsdb* command contains several error-checking routines to verify inode 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.

The *fsdb* command 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 inode 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 inodes
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 inode. 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 checks 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 inode, 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. Inodes are printed with labeled fields describing each element.

The following mnemonics are used for inode examination and refer to the current working inode:

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 inode format. This now becomes the current working inode.
ln=4	changes the link count for the working inode to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working inode.
2i.fd	prints the first 32 directory entries for the root inode of this file system.

d5i.fc	changes the current inode 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.
a2b.p0d	prints the third block of the current inode as directory entries.

SEE ALSO

fsck(1M).
dir(4), fs(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

fsstat – report file system status

SYNOPSIS

/etc/fsstat special_file

DESCRIPTION

The *fsstat* command reports on the status of the file system on *special_file*. During startup, this command is used to determine if the file system needs checking before it is mounted. The *fsstat* command succeeds if the file system is unmounted and appears okay. For the root file system, it succeeds if the file system is active and not marked bad.

SEE ALSO

fs(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

DIAGNOSTICS

The command has the following exit codes:

0 the file system is not mounted and appears okay, (except for root where 0 means mounted and okay).
1 the file system is not mounted and needs to be checked.
2 the file system is mounted.
3 the command failed.

NAME

fstyp – determine file system identifier

SYNOPSIS

fstyp *special*

DESCRIPTION

The *fstyp* command allows the user to determine the file system identifier of mounted or unmounted file systems using heuristic programs. The file system type is required by *mount*(2) and sometimes by *mount*(1M) to mount file systems of different types.

The directory **/etc/fstyp.d** contains a program for each file system type to be checked; each of these programs applies some appropriate heuristic to determine whether the supplied *special* file is of the type for which it checks. If it is, the program prints on standard output the usual file-system identifier for that type and exits with a return code of 0; otherwise it prints error messages on standard error and exits with a non-zero return code. *fstyp* runs the programs in **/etc/fstyp.d** in alphabetical order, passing *special* as an argument; if any program succeeds, its file-system type identifier is printed and *fstyp* exits immediately. If no program succeeds, *fstyp* prints "Unknown_fstyp" to indicate failure.

WARNING

The use of heuristics implies that the result of *fstyp* is not guaranteed to be accurate.

SEE ALSO

mount(1M).
mount(2), sysfs(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

ftape — interface to floppy tape drives

SYNOPSIS

ftape −SFViIot [−f *number*] [−d *description*] *device*

DESCRIPTION

ftape is the user interface to floppy tape drives. It uses ECC (Error Correction Codes) that go beyond the usual CRC checksums (which are implemented in the device driver) to allow the recovery of data that would otherwise be lost through tape errors. *ftape* can be used only with the floppy tape driver, *ft*(7). *device* must be the floppy tape driver device file, **/dev/ftape**. *ftape* is typically used with one of the archiving programs *cpio*(1) or *tar*(1).

The *ftape* program supports multiple *filesets* on a single cartridge, following the QIC-40 standard for the *Volume Table* (VTBL), and also supports multiple volume operations. When the program detects the end of the tape, it prompts for another cartridge. When reading multiple cartridge sets, it checks to make sure the volumes are inserted in the proper order.

The command line options are:

−i Read data from the tape and write it to *stdout*.

−I Same as **−i**, except that the program will not exit if uncorrectable errors occur.

−o Read data from *stdin* and write to it to the tape.

−t Read just the tape header and VTBL block, producing a table of contents of the tape volume.

−f Reads from or writes to fileset *number*. There can be as many as 64 filesets on an Irwin tape, and up to 116 on a QIC-40 tape. If no **−f** flag is given, the first fileset on the tape is assumed. When writing fileset number *N*, filesets 1 through *N*-1 must have been previously written, and any filesets numbered >*N* are destroyed.

−d *Description* is up to 44 characters of descriptive text. This text is stored in the VTBL header, and will be displayed by the **−t** flag, but has no effect on the data. The text should be quoted if it contains any special characters or blanks.

−S Specifies that a write servo operation is to be performed on the tape. The tape must be a new, unformatted tape or have been erased with a bulk tape eraser.

−F Specifies that a physical format operation is to be performed on the tape. When formatting a tape, there should be no other activity on the system.

−V Specifies that the tape is to be searched for bad blocks and a tape header placed at the beginning of the tape indicating the bad blocks on the tape.

EXAMPLES

To archive all files in the file system tree above the current directory into fileset 1 on the tape:

find . −print | cpio −o | ftape −o −d 'Monday backup' /dev/ftape

To restore all the files previously archived in fileset 1:

ftape −i /dev/ftape | cpio -i

SEE ALSO

ft(7).

DIAGNOSTICS

All diagnostic output, as well as the output from the **−t** flag, is directed to *stderr*. *stdout* is reserved for the data stream. Prompts for new tapes and responses to prompts are written and read from **/dev/tty**.

WARNINGS

A tape should only be servo written after being thoroughly erased with a bulk tape eraser or erratic behavior may result.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

fuser – identify processes using a file or file structure

SYNOPSIS

/etc/fuser [**–ku**] files | resources [**–**] [[**–ku**] files | resources]

DESCRIPTION

The *fuser* command outputs the process IDs of the processes that are using the *files* or remote *resources* specified as arguments. Each process ID is followed by a letter code, interpreted as follows: if the process is using the file as (1) its current directory, the code is **c**; (2) the parent of its current directory (only when the file is being used by the system), the code is **p**; or (3) its root directory, the code is **r**. For block-special devices with mounted file systems, all processes using any file on that device are listed. For remote resource names, all processes using any file associated with that remote resource (Remote File Sharing) are reported. (*fuser* cannot use the mount point of the remote resource; it must use the resource name.) For all other types of files (text files, executables, directories, devices, etc.), only the processes using that file are reported.

The following options may be used with *fuser*:

–u the user login name, in parentheses, also follows the process ID.

–k the SIGKILL signal is sent to each process. Since this option spawns kills for each process, the kill messages may not show up immediately [see *kill*(2)].

If more than one group of files are specified, the options may be respecified for each additional group of files. A lone dash cancels the options currently in force; then, the new set of options applies to the next group of files.

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.

You cannot list processes using a particular file from a remote resource mounted on your machine. You can only use the resource name as an argument.

Any user with permission to read **/dev/kmem** and **/dev/mem** can use *fuser*. Only the super-user can terminate another user's process

FILES

/unix	for system name list
/dev/kmem	for system image
/dev/mem	also for system image

SEE ALSO

mount(1M), ps(1).
kill(2), signal(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

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, *acctcon*(1M) 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 *acctcon*(1M) will fail when processing connect accounting records.

FILES

/etc/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), ed(1), runacct(1M).

acct(2), acct(4), utmp(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

getopt − parse command options

SYNOPSIS

set −− `getopt** optstring **$*`

DESCRIPTION

WARNING: Start using the new command *getopts*(1) in place of *getopt*(1). *getopt*(1) will not be supported in the next major release. For more information, see the **WARNINGS** section below.

The *getopt* command 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

getopts(1), sh(1).
getopt(3C) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

DIAGNOSTICS

The *getopt* command prints an error message on the standard error when it encounters an option letter not included in *optstring*.

WARNINGS

The *getopt*(1) command does not support the part of Rule 8 of the command syntax standard [see *intro*(1)] that permits groups of option-arguments following an option to be separated by white space and quoted. For example,

cmd −a −b −o "xxx z yy" file

is not handled correctly. To correct this deficiency, use the new command *getopts*(1) in place of *getopt*(1).

getopt(1) will not be supported in the next major release. For this release, a conversion tool has been provided, *getoptcvt*. For more information about *getopts* and *getoptcvt*, see the *getopts*(1) manual page.

If an option that takes an option-argument is followed by a value that is the same as one of the options listed in *optstring* (referring to the earlier EXAMPLE section, but using the following command line: **cmd -o -a file**), *getopt* will always treat **−a** as an option-argument to **−o**; it will never recognize **−a** as an option. For this case, the **for** loop in the example will shift past the *file* argument.

NAME

getopts, getoptcvt − parse command options

SYNOPSIS

getopts optstring name [arg ...]

/usr/lib/getoptcvt [−b] file

DESCRIPTION

The *getopts* command is used by shell procedures to parse positional parameters and to check for legal options. It supports all applicable rules of the command syntax standard [see Rules 3-10, *intro*(1)]. It should be used in place of the *getopt*(1) command. (See the **WARNING** below.)

optstring must contain the option letters the command using *getopts* will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, *getopts* will place the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell procedure is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, *getopts* places it in the shell variable **OPTARG**.

If an illegal option is encountered, **?** will be placed in *name*.

When the end of options is encountered, *getopts* exits with a non-zero exit status. The special option "−−" may be used to delimit the end of the options.

By default, *getopts* parses the positional parameters. If extra arguments (*arg* ...) are given on the *getopts* command line, *getopts* will parse them instead.

The */usr/lib/getoptcvt* command reads the shell script in *file*, converts it to use *getopts*(1) instead of *getopt*(1), and writes the results on the standard output.

−b the results of running */usr/lib/getoptcvt* will be portable to earlier releases of the UNIX system. */usr/lib/getoptcvt* modifies the shell script in *file* so that when the resulting shell script is executed, it determines at run time whether to invoke *getopts*(1) or *getopt*(1).

So all new commands will adhere to the command syntax standard described in *intro*(1), they should use *getopts*(1) or *getopt*(3C) to parse positional parameters and check for options that are legal for that command (see **WARNINGS** below).

EXAMPLE

The following fragment of a shell program 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 option-argument:

```
while getopts abo: c
do
        case $c in
        a | b)          FLAG=$c;;
        o)              OARG=$OPTARG;;
        \?)             echo $USAGE
                        exit 2;;
        esac
done
shift `expr $OPTIND − 1`
```

This code will accept any of the following as equivalent:

```
cmd −a −b −o "xxx z yy" file
cmd −a −b −o "xxx z yy" −− file
cmd −ab −o xxx,z,yy file
cmd −ab −o "xxx z yy" file
cmd −o xxx,z,yy −b −a file
```

SEE ALSO

intro(1), sh(1).

getopt(3C) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

WARNING

Although the following command syntax rule [see *intro*(1)] relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLE** section above, **a** and **b** are options, and the option **o** requires an option-argument:

cmd −aboxxx file	(Rule 5 violation: options with option-arguments must not be grouped with other options.)
cmd −ab −oxxx file	(Rule 6 violation: there must be white space after an option that takes an option-argument.)

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

DIAGNOSTICS

getopts 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 [**–h**] [**–t** timeout] [**–m** maparg] line
[speed [type [linedisc]]]

/etc/getty –c file

DESCRIPTION

The *getty* command 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. It can only be executed by the super-user; that is, a process with the user-ID of **root**. Initially, *getty* prints 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 the argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used. It does this by using the options and arguments specified.

The **–m** option specifies that the *ttymap*(1) command is to be run with the argument *maparg*. *ttymap* sets the terminal character mappings. Multiple argument words can be passed to *ttymap* by enclosing them in quotes. *ttymap* is run immediately before the prompt for the user *loginname*.

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 **–h** 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.

Speed, the optional second argument, 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.

Type, the optional third argument, is a character string describing to *getty* what type of terminal is connected to the line in question. *getty* recognizes the following types:

none	default
ds40-1	DATASPEED terminal 40/1
tektronix,tek	TEKTRONIX
vt61	Digital Equipment vt61
vt100	Digital Equipment vt100
hp45	Hewlett-Packard 45
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.

Linedisc, the optional fourth argument, 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**.

After the user's name has been typed in, it 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)].

The user's name is scanned to see if it contains any lower case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper case characters into the corresponding lower case characters.

Finally, *login* is **exec**'d 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.

Example

A valid *inittab* entry is:

00:23:respawn:/etc/getty −m /usr/lib/keyboard/french.map tty00 9600

FILES

/etc/gettydefs

SEE ALSO

ct(1C), init(1M), login(1), uugetty(1M), tty(7).
ioctl(2), gettydefs(4), inittab(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

BUGS

While *getty* understands simple single character quoting conventions, it is not possible to quote certain special control characters used by *getty*. Thus, you cannot log in via *getty* and type a **#**, **@**, **/**, **!**, **_**, backspace, **^U**, **^D**, or **&** as part of your login name or arguments. *getty* uses them to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. They will always be interpreted as having their special meaning.

WARNING

When connecting two computers using a direct connection, never invoke *getty*(1M) on the ports of both machines. Instead, use *uugetty*(1M).

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

graph − draw a graph

SYNOPSIS

graph [options]

DESCRIPTION

The *graph* command with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *tplot*(1G) filters.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes ", in which case they may be empty or contain blanks and numbers; labels never contain new-lines.

The following options are recognized, each as a separate argument:

−a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **−x**).

−b Break (disconnect) the graph after each label in the input.

−c Character string given by next argument is default label for each point.

−g Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

−l Next argument is label for graph.

−m Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (e.g., the Tektronix 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).

−s Save screen, do not erase before plotting.

−x [l] If **l** is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

−y [l] Similarly for *y*.

−h Next argument is fraction of space for height.

−w Similarly for width.

−r Next argument is fraction of space to move right before plotting.

−u Similarly to move up before plotting.

−t Transpose horizontal and vertical axes. (Option **−x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **−s** option is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

graphics(1G), spline(1G), tplot(1G).

BUGS

The *graph* command stores all points internally and drops those for which there is no room.

Segments that run out of bounds are dropped, not windowed.

Logarithmic axes may not be reversed.

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 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)]. Currently, the following *terminal*s are recognized:

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), term(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

grep – search a file for a pattern

SYNOPSIS

grep [options] limited regular expression [file ...]

DESCRIPTION

The *grep* command searches files for a pattern and prints all lines that contain that pattern. The *grep* command uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with *ed*(1) to match the patterns. It uses a compact non-deterministic algorithm.

Be careful using the characters **$**, *****, **[**, **^**, **|**, (,), and \ in the *limited regular expression* because they are also meaningful to the shell. It is safest to enclose the entire *limited regular expression* in single quotes '...'.

If no files are specified, *grep* assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

–b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
–c Print only a count of the lines that contain the pattern.
–i Ignore upper/lower case distinction during comparisons.
–h Prevents the name of the file containing the matching line from being appended to that line. Used when searching multiple files.
–l Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.
–n Precede each line by its line number in the file (first line is 1).
–s Suppress error messages about nonexistent or unreadable files.
–v Print all lines except those that contain the pattern.

SEE ALSO

ed(1), egrep(1), fgrep(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

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**, which is included as part of the basic software development set.

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.

NAME

hd − display files in hexadecimal format

SYNOPSIS

hd [−*format* [**−s** *offset*] [**−n** *count*] [*file*] ...

DESCRIPTION

The *hd* command displays the contents of files in hexadecimal, octal, decimal, and character formats. Control over the specification of ranges of characters is also available. The default behavior is with the following flags set: **−abx −A**. This says that addresses (file offsets) and bytes are printed in hexadecimal and that characters are also printed. If no *file* argument is given, the standard input is read.

Options include:

−s *offset* Specify the beginning offset in the file where printing is to begin. If no *file* argument is given or if a seek fails because the input is a pipe, *offset* bytes are read from the input and discarded. Otherwise, a seek error will terminate processing of the current file.

The *offset* may be given in decimal, hexadecimal (preceded by **0x**), or octal (preceded by a **0**). It is optionally followed by one of the following multipliers: **w**, **l**, **b**, or **k**; for words (2 bytes), long words (4 bytes), blocks (512 bytes), or K bytes (1024 bytes). Note that this is the one case where **b** does *not* stand for bytes. Since specifying a hexadecimal offset in blocks would result in an ambiguous trailing **b**, any offset and multiplier may be separated by an asterisk (*).

−n *count* Specify the number of bytes to process. The *count* is in the same format as *offset* above.

FORMAT FLAGS

Format flags may specify addresses, characters, bytes, words (2 bytes), or longs (4 bytes) to be printed in hexadecimal, decimal, or octal. Two special formats may also be indicated: text or ASCII. Format and base specifiers may be freely combined and repeated as desired to specify different bases (hexadecimal, decimal, or octal) for different output formats (addresses, characters, etc.). All format flags appearing in a single argument are applied as appropriate to all other flags in that argument.

acbwlA Output format specifiers for addresses, characters, bytes, words, longs, and ASCII, respectively. Only one base specifier will be used for addresses; the address will appear on the first line of output that begins each new offset in the input.

The character format prints printable characters unchanged special C escapes as defined in the language, and remaining values in the specified base.

The ASCII format prints all printable characters unchanged, and all others as a period (.). This format appears to the right of the first of other specified output formats. A base specifier has no meaning with the ASCII format. If no other output format (other than addresses) is given, **bx** is assumed. If no base specifier is given, *all* of **xdo** are used.

xdo Output base specifiers for hexadecimal, decimal, and octal. If no format specifier is given, *all* of **acbwl** are used.

t Print a text file, each line preceded by the address in the file. Normally, lines should be terminated by a **\n** character, but long lines will be broken up. Control characters in the range 0x00 to 0x1f are printed as '^@' to '^_'. Bytes with the high bit set are preceded by a tilde (~) and printed as if the high bit were not set. The special characters (^, ~, \) are preceded by a backslash (\) to escape their special meaning. As special cases, two values are represented numerically as '\177' and '\377'. This flag will override all output format specifiers except addresses.

NAME

hp — handle special functions of Hewlett-Packard 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 usage is in conjunction with DOCUMENTER'S WORKBENCH Software:

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 Underlined 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.

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), greek(1).

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.

NAME

id — print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

The *id* command outputs the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs are different, both are printed.

SEE ALSO

logname(1).
getuid(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

idbuild — build new UNIX System kernel

SYNOPSIS

/etc/conf/bin/idbuild

DESCRIPTION

This script builds a new UNIX System kernel using the current system configuration in **/etc/conf/**. Kernel reconfigurations are usually done after a device driver is installed or after system tunable parameters are modified. The script uses the shell variable **$ROOT** from the user's environment as its starting path. Except for the special case of kernel development in a non-root source tree, the shell variable **$ROOT** should always be set to null or to /. *idbuild* exits with a return code of zero on success and nonzero on failure.

Building a new UNIX System image consists of generating new system configuration files, then link-editing the kernel and device driver object modules in the **etc/conf/pack.d** object tree. This is done by *idbuild* by calling the following commands:

etc/conf/bin/idconfig To build kernel configuration files.

etc/conf/bin/idmkunix To process the configuration files and link-edit a new UNIX System image.

The system configuration files are built by processing the Master and System files representing device driver and tunable parameter specifications. For the INTERACTIVE UNIX Operating System, the files **etc/conf/cf.d/mdevice** and **etc/conf/cf.d/mtune** represent the "Master" information. The file **etc/conf/cf.d/stune** and the files specified in **etc/conf/sdevice.d/*** represent the "System" information. The kernel also has file system type information defined in the files specified by **etc/conf/sfsys.d/*** and **etc/conf/mfsys.d/***. The file **/etc/conf/cf.d/defines** and the files specified in **/etc/conf/define.d** represent the preprocessor define symbols needed to compile and link the kernel.

Once a new UNIX System kernel has been configured, a lock file is set in **etc/.new_unix** which causes the new kernel to replace **/unix** on the next system shutdown (i.e., on the next entry to the *init 0* state). Upon the next system boot, the new kernel will be executed.

The command line options are:

−r root_dir
Set environment variable **$ROOT** to **root_dir** for the build command. This overrides the current value of **$ROOT**.

−k *suffix*
Build the kernel in a manner consistent with the *kconfig* command. The following differences will be noted:

The output of *idconfig* will be left in the **$ROOT/etc/conf/tmp** directory.

The lock file **/etc/.new_unix** is not set.

The new kernel is named **unix.***suffix*.

The resulting kernel and configuration files are saved in **$ROOT/etc/conf/kconfig.d/unix.*suffix*.d**.

This option is intended for use by the *kconfig* command.

Error Messages

Since *idbuild* calls other system commands to accomplish system reconfiguration and link editing, it will report all errors encountered by those commands, then clean up intermediate files created in the process. In general, the exit value 1 indicates an error was encountered by *idbuild*.

The errors encountered fall into the following categories:

Master file error messages
System file error messages
Tunable file error messages
Compiler and Link-editor error messages

All error messages are designed to be self-explanatory.

SEE ALSO

idconfig(1M), idinstall(1M), idmkunix(1M), idtune(1M).
mdevice(4), mfsys(4), mtune(4), sdevice(4), sfsys(4), stune(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

idcheck – returns selected information

SYNOPSIS

/etc/conf/bin/idcheck

DESCRIPTION

This command returns selected information about the system configuration. It is useful in add-on device Driver Software Package (DSP) installation scripts to determine if a particular device driver has already been installed, or to verify that a particular interrupt vector, I/O address or other selectable parameter is in fact available for use. The various forms are:

idcheck –p device-name [–i dir] [–r]

idcheck –v vector [–i dir] [–r]

idcheck –d dma-channel [–i dir] [–r]

idcheck –a –l lower_address –u upper_address [–i dir] [–r]

idcheck –c –l lower_address –u upper_address [–i dir] [–r]

This command scans the System and Master modules and returns:

100 if an error occurs.

0 if no conflict exists.

a positive number greater than 0 and less than 100 if a conflict exists.

The command line options are:

–r Report device name of any conflicting device on stdout.

–p *device-name* This option checks for the existence of four different components of the DSP. The exit code is the addition of the return codes from the four checks.

Add 1 to the exit code if the DSP directory under **/etc/conf/pack.d** exists.

Add 2 to the exit code if the Master module has been installed.

Add 4 to the exit code if the System module has been installed.

Add 8 to the exit code if the Kernel was built with the System module.

Add 16 to the exit code if a Driver.o is part of the DSP (vs. a stubs.c file).

–v *vector* Returns 'type' field of device that is using the vector specified (i.e., another DSP is already using the vector).

–d *dma-channel* Returns 1 if the dma channel specified is being used.

–a This option checks whether the IOA region bounded by "lower" and "upper" conflict with another DSP ("lower" and "upper" are specified with the **–l** and

−u options). The exit code is the addition of two different return codes.

Add 1 to the exit code if the IOA region overlaps with another device.

Add 2 to the exit code if the IOA region overlaps with another device and that device has the 'O' option specified in the *type* field of the Master module. The 'O' option permits a driver to overlap the IOA region of another driver.

−c Returns 1 if the CMA region bounded by "lower" and "upper" conflict with another DSP ("lower" and "upper" are specified with the **−l** and **−u** options).

−l *address* Lower bound of address range specified in hex. The leading 0x is unnecessary.

−u *address* Upper bound of address range specified in hex. The leading 0x is unnecessary.

−i *dir* Specifies the directory in which the ID files *sdevice* and *mdevice* reside. The default directory is **/etc/conf/cf.d**.

ERROR MESSAGES

There are no error messages or checks for valid arguments to options. *idcheck* interprets these arguments using the rules of *scanf*(3) and queries the *sdevice* and *mdevice* files. For example, if a letter is used in the place of a digit, *scanf*(3) will translate the letter to 0. *idcheck* will then use this value in its query.

SEE ALSO

idinstall(1M).

mdevice(4), sdevice(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

idconfig — produce a new kernel configuration

SYNOPSIS

/etc/conf/bin/idconfig

DESCRIPTION

The *idconfig* command takes as its input a collection of files specifying the configuration of the next INTERACTIVE UNIX Operating System to be built. A collection of output files for use by *idmkunix* is produced.

The input files expected by *idconfig* are as follows:

cf.d/mdevice	Master device specifications
cf.d/sdevice	System device specifications
cf.d/mtune	Master parameter specifications
cf.d/stune	System parameter specifications
cf.d/mfsys	File system type master data
defines.d/*	Preprocessor define symbols for individual modules
cf.d/sfsys	File system type system data
cf.d/sassign	Device assignment file

The output files produced by *idconfig* are as follows:

cf.d/conf.c	Kernel data structures and function definitions
cf.d/config.h	Kernel parameter and device definitions
cf.d/vector.c	Interrupt vector definitions
cf.d/direct	List of all driver components included in the build
cf.d/fsconf.c	File system type configuration data
cf.d/defines	List of all preprocessor define symbols needed to compile the kernel sources (**space.c** files)

The command line options are as follows:

−o *directory*	Output files will be created in the directory specified rather than **/etc/conf/cf.d**.
−i *directory*	Input files that normally reside in **/etc/conf/cf.d** can be found in the directory specified.
−r *directory*	The directory specified will be used as the ID "root" directory rather than **/etc/conf**.
−d *file*	Use *file* name rather than **sdevice** for input.
−t *file*	Use *file* name rather than **stune** for input.
−T *file*	Use *file* name rather than **mtune** for input.
−a *file*	Use *file* name rather than **sassign** for input.
−c *file*	Redirect **conf.c** output to *file* name.
−h *file*	Redirect **config.h** output to *file* name.
−v *file*	Redirect **vector.c** output to *file* name.
−p *file*	Redirect **direct** output to *file* name.
−D, −m, −s	These options are no longer supported.
−#	Print debugging information.
−z *file*	Redirect **defines** output to *file* name.

ERROR MESSAGES

An exit value of zero indicates success. If an error was encountered, *idconfig* will exit with a nonzero value and report an error message. All error messages are designed to be self-explanatory.

SEE ALSO

idbuild(1M), idinstall(1M), idmkunix(1M).
mdevice(4), mtune(4), sdevice(4), stune(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

idinstall − add, delete, update, or get device driver configuration data

SYNOPSIS

/etc/conf/bin/idinstall −[adug] [−e] −[msoptnirhcl] −Rdir dev_name

DESCRIPTION

The *idinstall* command is called by a Driver Software Package (DSP) Install script or Remove script to Add (**−a**), Delete (**−d**), Update (**−u**), or Get (**−g**) device driver configuration data. *idinstall* expects to find driver component files in the current directory. When components are installed or updated, they are moved or appended to files in the **/etc/conf** directory and then deleted from the current directory unless the **−k** flag is used. The options for the command are as follows:

Action Specifiers:

- **−a** Add the DSP components
- **−d** Remove the DSP components
- **−u** Update the DSP components
- **−g** Get the DSP components (print to **std out**, except Master)

Component Specifiers (if no component is specified, the default is all options except for the **−g** option, where a single component must be specified explicitly):

- **−m** Master component
- **−s** System component
- **−o** Driver.o component
- **−p** Space.c component
- **−t** Stubs.c component
- **−n** Node (special file) component
- **−i** Inittab component
- **−r** Device Initialization (rc) component
- **−h** Device shutdown (sd) component
- **−c** Mfsys component: file system type config (Master) data
- **−l** Sfsys component: file system type local (System) data
- **−z** Define component: list of preprocessor symbols needed to compile this module

Miscellaneous:

- **−e** Disable free disk space check
- **−k** Keep files (do not remove from current directory) upon add or update.

In the simplest case of installing a new DSP, the command syntax used by the DSP's Install script should be:

idinstall −a *dev_name*

In this case the command will require and install a Driver.o, Master and System entry, and optionally install the Space.c, Stubs.c, Node, Init, Rc, Shutdown, Mfsys, and Sfsys components if those modules are present in the current directory.

The **Driver.o**, **Space.c**, and **Stubs.c** files are moved to a directory in **/etc/conf/pack.d**. The *dev_name* is passed as an argument, which is used as the directory name. The remaining components are stored in the corresponding directories under **/etc/conf** in a file whose name is **dev_name**. For example, the Node file would be moved to **/etc/conf/node.d/dev_name**.

The *idinstall* **−m** usage provides an interface to the *idmaster* command which will add, delete, and update *mdevice* file entries using a Master file from the local directory. An interface is provided here so that driver writers have a consistent interface to install any DSP component.

As stated above, driver writers will generally use only the *idinstall* **−a** *dev_name* form of the command. Other options of *idinstall* are provided to allow an update DSP (i.e., one that replaces an existing device driver component) to be installed, and to support installation of multiple controller boards of the same type.

If the call to *idinstall* uses the **−u** (update) option, it will:

Overlay the files of the old DSP with the files of the new DSP.

Invoke the *idmaster* command with the "update" option if a Master module is part of the new DSP.

idinstall also does a verification that enough free disk space is available to start the reconfiguration process. This is done by calling the *idspace* command. *idinstall* will fail if insufficient space exists, and exit with a nonzero return code. The **−e** option bypasses this check.

idinstall makes a record of the last device installed in a file (**/etc/.last_dev_add**) and saves all removed files from the last delete operation in a directory (**/etc/.last_dev_del**). These files are recovered by **/etc/conf/bin/idmkenv** whenever it is determined that a system reconfiguration was aborted due to a power failure or unexpected system reboot.

ERROR MESSAGES

An exit value of zero indicates success. If an error was encountered, *idinstall* will exit with a non-zero value and report an error message. All error messages are designed to be self-explanatory. Typical error messages that can be generated by *idinstall* are as follows:

Device package already exists.
Cannot make the driver package directory.
Cannot remove driver package directory.
Local directory does not contain a Driver object (**Driver.o**) file.
Local directory does not contain a Master file.
Local directory does not contain a System file.
Cannot remove driver entry.

SEE ALSO

idspace(1M), idcheck(1M).
mdevice(4), sdevice(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

idmkinit – read files containing specifications

SYNOPSIS

/etc/conf/bin/idmkinit

DESCRIPTION

This command reads the files containing specifications of **/etc/inittab** entries from **/etc/conf/init.d** and constructs a new *inittab* file in **/etc/conf/cf.d**. It returns 0 on success and a positive number on error.

The files in **/etc/conf/init.d** are copies of the *Init* modules in device Driver Software Packages (DSP). There is at most one *Init* file per DSP. Each file contains one line for each *inittab* entry to be installed. There may be multiple lines (i.e., multiple *inittab* entries) per file. An *inittab* entry has the form:

id:rstate:action:process

(The *id* field in the above example is often called the *tag*.)

The *Init* module entry must have one of the following forms:

action:process

rstate:action:process

id:rstate:action:process

When *idmkinit* encounters an entry of the first type, a valid *id* field will be generated, and an *rstate* field of 2 (indicating run on "*init*state 2") will be generated. When an entry of the second type is encountered, only the *id* field is prefixed. An entry of the third type is incorporated into the new *inittab* unchanged.

Since add-on *inittab* entries specify *init* state 2 for their *rstate* field most often, an entry of the first type should almost always be used. An entry of the second type may be specified if you need to specify other than state 2. DSPs should avoid specifying the *id* field as in the third entry since other add-on applications or DSPs may have already used the *id* value you have chosen. The **/etc/init** program will encounter serious errors if one or more *inittab* entries contain the same *id* field.

idmkinit determines which of the three forms above is being used for the entry by requiring each entry to have a valid **action** keyword. Valid **action** values are as follows:

off
respawn
ondemand
once
wait
boot
bootwait
powerfail
powerwait
initdefault
sysinit

The *idmkinit* command is called automatically upon entering *init* state 2 on the next system reboot after a kernel reconfiguration to establish the correct **/etc/inittab** for the running **/unix** kernel. *idmkinit* can be called as a user-level command to test modification of *inittab* before a DSP is actually built. It is also useful in installation scripts that do not reconfigure the kernel but need to create *inittab* entries. In this case, the *inittab* generated by *idmkinit* must be copied to **/etc/inittab**, and a *telinit q* command must be run to make the new entry take effect.

The command line options are

−o *directory* *inittab* will be created in the directory specified rather than **/etc/conf/cf.d**.

−i *directory* The ID file **init.base**, which normally resides in **/etc/conf/cf.d**, can be found in the directory specified.

−e *directory* The *Init* modules that are usually in **/etc/conf/init.d** can be found in the directory specified.

−# Print debugging information.

Enhanced Functionality

Files in **/etc/conf/init.d** that have the same name as a DSP driver will only be included in **/etc/inittab** if the driver is configured. All of the other files in **/etc/conf/init.d** will always be included.

EXAMPLE

/etc/conf/init.d/foo will always be included in **/etc/inittab**, providing there is no driver named *foo*.

/etc/conf/init.d/asy will only be included in **/etc/inittab** if *asy* is configured in the **sdevice** file.

ERROR MESSAGES

An exit value of zero indicates success. If an error was encountered, *idmkinit* will exit with a nonzero value and report an error message. All error messages are designed to be self-explanatory.

SEE ALSO

idbuild(1), idinstall(1M), idmknod(1M), init(1M).
inittab(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

idmknod – removes nodes and reads specifications of nodes

SYNOPSIS

/etc/conf/bin/idmknod

DESCRIPTION

This command performs the following functions:

> Removes the nodes for non-required devices (those that do not have an 'r' in field 3 of the the device's *mdevice* entry) from **/dev**. Ordinary files will not be removed. If the **/dev** directory contains subdirectories, those subdirectories will be transversed and nodes found for non-required devices will be removed as well. If empty subdirectories result due to the removal of nodes, the subdirectories are then removed.
>
> Reads the specifications of nodes given in the files contained in **/etc/conf/node.d** and installs these nodes in **/dev**. If the node specification defines a path containing subdirectories, the subdirectories will be made automatically.
>
> Returns 0 on success and a positive number on error.

The *idmknod* command is run automatically upon entering init state 2 on the next system reboot after a kernel reconfiguration to establish the correct representation of device nodes in the **/dev** directory for the running **/unix** kernel. *idmknod* can be called as a user level command to test modification of the **/dev** directory before a DSP is actually built. It is also useful in installation scripts that do not reconfigure the kernel, but need to create **/dev** entries.

The files in **/etc/conf/node.d** are copies of the *Node* modules installed by device Driver Software Packages (DSP). There is at most one file per DSP. Each file contains one line for each node that is to be installed. The format of each line is:

> Name of device entry (field 1) in the *mdevice* file (The *mdevice* entry will be the line installed by the DSP from its *Master* module). This field must be from 1 to 8 characters in length. The first character must be a letter. The others may be letters, digits, or underscores.
>
> Name of node to be inserted in **/dev.** The first character must be a letter. The others may be letters, digits, or underscores. This field can be a path relative to **/dev**, and *idmknod* will create subdirectories as needed.
>
> The character **b** or **c**. A **b** indicates that the node is a 'block' type device and **c** indicates 'character' type device.
>
> Minor device number. This value must be between 0 and 255. If this field is a non-numeric, it is assumed to be a request for a streams clone device node, and *idmknod* will set the minor number to the value of the major number of the device specified.

Some example node file entries are as follows:

asy tty00 c 1
makes /dev/tty00 for device 'asy' using minor device 1.

qt rmt/c0s0 c 4
makes /dev/rmt/c0s0 for device 'qt' using minor device 4.

clone net/nau/clone c nau
makes /dev/net/nau/clone for device 'clone'. The minor device number is set to the major device number of device 'nau'.

The command line options are:

−o *directory* Nodes will be installed in the directory specified rather than **/dev**.

−i *directory* The file *mdevice* which normally resides in **/etc/conf/cf.d**, can be found in the directory specified.

−e *directory* The *Node* modules that normally reside in **/etc/conf/node.d** can be found in the directory specified.

−s Suppress removing nodes (just add new nodes).

ERROR MESSAGES

An exit value of zero indicates success. If an error was encountered due to a syntax or format error in a *node* entry, an advisory message will be printed to *stdout* and the command will continue. If a serious error is encountered (i.e., a required file cannot be found), *idmknod* will exit with a non-zero value and report an error message. All error messages are designed to be self-explanatory.

SEE ALSO

idinstall(1M), idmkinit(1M).

mdevice(4), sdevice(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

idmkunix – build new UNIX system kernel

SYNOPSIS

/etc/conf/bin/idmkunix

DESCRIPTION

The *idmkunix* command creates a bootable INTERACTIVE UNIX Operating System kernel in the directory **/etc/conf/cf.d**. The component kernel "core" files and device driver object files contained in subdirectories of **/etc/conf/pack.d** are used as input along with device and parameter definition files produced by *idconfig*. In brief, the required input files are:

/etc/conf/cf.d/conf.c	Kernel data structures and function definitions
/etc/conf/cf.d/config.h	Kernel parameter and device definitions
/etc/conf/cf.d/vector.c	Interrupt vector definitions
/etc/conf/cf.d/direct	Listing of all driver components included in the build
/etc/conf/cf.d/fsconf.c	File system type configuration data
/etc/conf/cf.d/defines	List of all preprocessor define symbols needed to compile kernel source files
/etc/conf/cf.d/vuifile	Memory management definitions for the kernel
/etc/conf/pack.d/*/Driver.o	Component kernel object files
/etc/conf/pack.d/*/space.c	Component kernel space allocation files
/etc/conf/pack.d/*/stubs.c	Component kernel stubs files

The command line options are as follows:

–o *directory* The file **unix** be created in the directory specified rather than **/etc/conf/cf.d.**

–i *directory* Input files that normally reside in **/etc/conf/cf.d** can be found in the directory specified.

–r *directory* The directory specified will be used as the ID "root" directory rather than **/etc/conf**.

–c, cc, –l, ld These options are no longer supported.

–# Print debugging information.

ERROR MESSAGES

An exit value of zero indicates success. If an error was encountered, *idmkunix* will exit with a nonzero value and report an error message. All error messages are designed to be self-explanatory.

SEE ALSO

idbuild(1M), idconfig(1M), idinstall(1M).
mdevice(4), mtune(4), sdevice(4), stune(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

idspace — investigates free space

SYNOPSIS

/etc/conf/bin/idspace [**−i** inodes] [**−r** blocks] [**−u** blocks] [**−t** blocks]

DESCRIPTION

This command investigates free space in /, **/usr**, and **/tmp** file systems to determine whether sufficient disk blocks and inodes exist in each of potentially 3 file systems. The default tests that *idspace* performs are as follows:

> Verify that the root file system (/) has 400 blocks more than the size of the current **/unix**. This verifies that a device driver being added to the current **/unix** can be built and placed in the root directory. A check is also made to insure that 100 inodes exist in the root directory.
>
> Determine whether a **/usr** file system exists. If it does exist, a test is made that 400 free blocks and 100 inodes are available in that file system. If the file system does not exist, *idspace* does not complain since files created in **/usr** by the reconfiguration process will be created in the root file system and space requirements are covered by the test in (1.) above.
>
> Determine whether a **/tmp** file system exists. If it does exist, a test is made that 400 free blocks and 100 inodes are available in that file system. If the file system does not exist, *idspace* does not complain since files created in **/tmp** by the reconfiguration process will be created in the root file system and space requirements are covered by the test in (1.) above.

The command line options are:

−i *inodes* This option overrides the default test for 100 inode in all of the *idspace* checks.

−r *blocks* This option overrides the default test for **/unix** size + 400 blocks when checking the root (/) file system. When the **−r** option is used, the **/usr** and **/tmp** file systems are not tested unless explicitly specified.

−u *blocks* This option overrides the default test for 400 blocks when checking the **/usr** file system. When the **−u** option is used, the root (/) and **/tmp** file systems are not tested unless explicitly specified. If **/usr** is not a separate file system, an error is reported.

−t *blocks* This option overrides the default test for 400 blocks when checking the **/tmp** file system. When the **−t** option is used, the root (/) and **/usr** file systems are not tested unless explicitly specified. If **/tmp** is not a separate file system, an error is reported.

ERROR MESSAGES

An exit value of zero indicates success. If insufficient space exists in a file system or an error was encountered due to a syntax or format error, *idspace* will report a message. All error messages are designed to be self-explanatory. The specific exit values are as follows:

0 success.

1 command syntax error, or needed file does not exist.

2 file system has insufficient space or inodes.

3 requested file system does not exist (**−u** and **−t** options only).

SEE ALSO

idbuild(1M), idinstall(1M).

NAME

idtune − attempts to set value of a tunable parameter

SYNOPSIS

/etc/conf/bin/idtune [−f | −m] name value

DESCRIPTION

This script attempts to set the value of a tunable parameter. The tunable parameter to be changed is indicated by *name*. The desired value for the tunable parameter is *value*.

If there is already a value for this parameter (in the **stune** file), the user will normally be asked to confirm the change with the following message:

> Tunable Parameter *name* is currently set to *old_value*.
> Is it OK to change it to *value*? (y/n)

If the user answers **y**, the change will be made. Otherwise, the tunable parameter will not be changed, and the following message will be displayed:

> *name* left at *old_value*.

However, if the **−f** (force) option is used, the change will always be made and no messages will ever be given.

If the **−m** (minimum) option is used and there is an existing value which is greater than the desired value, no change will be made and no message will be given.

If system tunable parameters are being modified as part of a device driver or application add-on package, it may not be desirable to prompt the user with the above question. The add-on package Install script may chose to override the existing value using the **−f** or **−m** options. However, care must be taken not to invalidate a tunable parameter modified earlier by the user or another add-on package.

In order for the change in parameter to become effective, the UNIX system kernel must be rebuilt and the system rebooted.

DIAGNOSTICS

The exit status will ne non-zero if errors are encountered.

SEE ALSO

idbuild(1).

mtune(4), stune(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

infocmp — compare or print out terminfo descriptions

SYNOPSIS

infocmp [**−d**] [**−c**] [**−n**] [**−I**] [**−L**] [**−C**] [**−r**] [**−u**] [**−s d|i|l|c**] [**−v**] [**−V**] [**−1**] [**−w** width] [**−A** directory] [**−B** directory] [termname ...]

DESCRIPTION

The *infocmp* command can be used to compare a binary *terminfo*(4) entry with other terminfo entries, rewrite a *terminfo*(4) description to take advantage of the **use=** terminfo field, or print out a *terminfo*(4) description from the binary file [*term*(4)] in a variety of formats. In all cases, the Boolean fields will be printed first, followed by the numeric fields, followed by the string fields.

Default Options

If no options are specified and zero or one *termnames* are specified, the **−I** option will be assumed. If more than one *termname* is specified, the **−d** option will be assumed.

Comparison Options [−d] [−c] [−n]

The *infocmp* command compares the *terminfo*(4) description of the first terminal *termname* with each of the descriptions given by the entries for the other terminal's *termnames*. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: **F** for boolean variables, **−1** for integer variables, and **NULL** for string variables.

−d produce a list of each capability that is different. In this manner, if one has two entries for the same terminal or similar terminals, using *infocmp* will show what is different between the two entries. This is sometimes necessary when more than one person produces an entry for the same terminal and one wants to see what is different between the two.

−c produce a list of each capability that is common between the two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **−u** option is worth using.

−n produce a list of each capability that is in neither entry. If no *termnames* are given, the environment variable **TERM** will be used for both of the *termnames*. This can be used as a quick check to see if anything was left out of the description.

Source Listing Options [−I] [−L] [−C] [−r]

The **−I**, **−L**, and **−C** options will produce a source listing for each terminal named.

−I use the *terminfo*(4) names

−L use the long C variable name listed in **<term.h>**

−C use the *termcap* names

−r when using **−C**, put out all capabilities in *termcap* form

If no *termnames* are given, the environment variable **TERM** will be used for the terminal name.

The source produced by the −C option may be used directly as a *termcap* entry, but not all of the parameterized strings may be changed to the *termcap* format. *infocmp* will attempt to convert most of the parameterized information, but that which it doesn't will be plainly marked in the output and commented out. These should be edited by hand.

All padding information for strings will be collected together and placed at the beginning of the string where *termcap* expects it. Mandatory padding (padding information with a trailing '/') will become optional.

All *termcap* variables no longer supported by *terminfo*(4), but which are derivable from other *terminfo*(4) variables, will be output. Not all *terminfo*(4) capabilities will be translated; only those variables which were part of *termcap* will normally be output. Specifying the **−r** option will take off this restriction, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output, mandatory padding is not supported, and *termcap* strings were not as flexible; it is not always possible to convert a *terminfo*(4) string capability into an equivalent *termcap* format. Not all of these strings will be able to be converted. A subsequent conversion of the *termcap* file back into *terminfo*(4) format will not necessarily reproduce the original *terminfo*(4) source.

Some common *terminfo* parameter sequences, their *termcap* equivalents, and some terminal types which commonly have such sequences are:

Terminfo	Termcap	Representative Terminals
%p1%c	%.	adm
%p1%d	%d	hp, ANSI standard, vt100
%p1%'x'%+%c	%+x	concept
%i	%i	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	%>xy	concept
%p2 is printed before %p1	%r	hp

Use= Option [−u]

−u produce a *terminfo*(4) source description of the first terminal *termname* which is relative to the sum of the descriptions given by the entries for the other terminals' *termnames*. It does this by analyzing the differences between the first *termname* and the other *termnames* and producing a description with **use=** fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using *infocmp* will show what can be done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first *termname*, but one of the other *termname* entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries that has this capability gives a different value for the capability than that in the first *termname*.

The order of the other *termname* entries is significant. Since the terminfo compiler *tic*(1M) does a left-to-right scan of the capabilities, specifying two **use=** entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given. *infocmp* will flag any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability *after* a **use=** entry that contains that capability will cause the second specification to be ignored. Using *infocmp* to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation time, is specifying extra **use=** fields that are superfluous. *infocmp* will flag any other *termname* **use=** fields that were not needed.

Other Options [−s d|i|l|c] [−v] [−V] [−1] [−w width]

−s sort the fields within each type according to the argument below:

- **d** leave fields in the order that they are stored in the *terminfo* data base.
- **i** sort by *terminfo* name.
- **l** sort by the long C variable name.
- **c** sort by the *termcap* name.

If no **−s** option is given, the fields printed out will be sorted alphabetically by the *terminfo* name within each type, except in the case of the **−C** or the **−L** options, which cause the sorting to be done by the *termcap* name or the long C variable name, respectively.

−v print out tracing information on standard error as the program runs.

−V print out the version of the program in use on standard error and exit.

−1 cause the fields to print out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.

−w change the output to *width* characters.

Changing Data Bases [−A directory] [−B directory]

The location of the compiled *terminfo*(4) data base is taken from the environment variable **TERMINFO**. If the variable is not defined or the terminal is not found in that location, the system *terminfo*(4) data base, usually in **/usr/lib/terminfo**, will be used. The options **−A** and **−B** may be used to override this location. The **−A** option will set **TERMINFO** for the first *termname* and the **−B** option will set **TERMINFO** for the other *termnames*. With this, it is possible to compare descriptions for a terminal with the same name located in two different data bases. This is useful for comparing descriptions for the same terminal created by different people. Otherwise the terminals would have to be named differently in the *terminfo*(4) data base for a comparison to be made.

FILES

/usr/lib/terminfo/?/* compiled terminal description data base

DIAGNOSTICS

malloc is out of space!
: There was not enough memory available to process all the terminal descriptions requested. Run *infocmp* several times, each time including a subset of the desired *termnames*.

use= order dependency found:
: A value specified in one relative terminal specification was different from that in another relative terminal specification.

'use=term' did not add anything to the description.
: A relative terminal name did not contribute anything to the final description.

must have at least two terminal names for a comparison to be done.
: The **−u**, **−d**, and **−c** options require at least two terminal names.

SEE ALSO

captoinfo(1M).
tic(1M), curses(3X), term(4), terminfo(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.
Chapter 10 of the *Programmer's Guide*.

NOTE

The *termcap* data base (from earlier releases of UNIX System V) may not be supplied in future releases.

NAME

init, telinit – process control initialization

SYNOPSIS

/etc/init [0123456SsQqabc]

/etc/telinit [0123456SsQqabc]

DESCRIPTION

Init

init is a general process spawner. Its primary role is to create processes from information stored in the file **/etc/inittab** [see *inittab*(4)].

At any given time, the system is in one of eight possible run levels. A run level is a software configuration of the system under which only a selected group of processes exist. The processes spawned by *init* for each of these run levels is defined in **/etc/inittab**. *init* can be in one of eight run levels, **0–6** and S or **s** (run levels S and **s** are identical). The run level changes when a privileged user runs **/etc/init**. This user-spawned *init* sends appropriate signals to the original *init* spawned by the operating system when the system was booted, telling it which run level to change to.

The following are the arguments to *init*:

0 shut the machine down so it is safe to remove the power. Have the machine remove power if it can. This state can be executed only from the console.

1 put the system in single-user mode. Unmount all file systems except root. All user processes are killed except those connected to the console. This state can be executed only from the console.

2 put the system in multiuser mode. All multiuser environment terminal processes and daemons are spawned. This state is commonly referred to as the multiuser state.

3 start the remote file sharing processes and daemons. Mount and advertise remote resources. Run level 3 extends multiuser mode and is known as the remote-file-sharing state.

4 is available to be defined as an alternative multiuser environment configuration. It is not necessary for system operation and is usually not used.

5 Stop the UNIX system and go to the firmware monitor.

6 Stop the UNIX system and reboot to the state defined by the **initdefault** entry in **/etc/inittab**.

a,b,c process only those **/etc/inittab** entries having the **a, b** or **c** run level set. These are pseudo-states, which may be defined to run certain commands, but which do not cause the current run level to change.

Q,q re-examine **/etc/inittab**.

S,s enter single-user mode. When this occurs, the terminal which executed this command becomes the system console. This is the only run level that doesn't require the existence of a properly formatted **/etc/inittab** file. If this file does not exist, then by default the only legal run level that *init* can enter is the single-user mode. When the system enters **S** or **s**, all mounted file systems remain mounted and only processes spawned by *init* are killed.

When a UNIX system is booted, *init* is invoked and the following occurs. First, *init* looks in **/etc/inittab** for the **initdefault** entry [see *inittab*(4)]. If there is one, *init* uses the run level specified in that entry as the initial run level to enter. If there is no **initdefault** entry in **/etc/inittab**, *init* requests that the user enter a run level from the virtual system console. If an **S** or **s** is entered, *init* goes to the single-user state. In the single-user state, the virtual console terminal is assigned to the user's terminal and is opened for reading and writing. The *sulogin* command, which requires the user to enter the root password, is invoked and a message is generated on the physical console saying where the virtual console has been relocated. Use either *init* or *telinit* to signal *init* to change the run level of the system. Note that if the shell is terminated (via an end-of-file), *init* will only re-initialize to the single-user state if the **/etc/inittab** file does not exist.

If a **0** through **6** is entered, *init* enters the corresponding run level. Note that, on the 80386 computer, the run levels **0**, **1**, **5**, and **6** are reserved states for shutting the system down; the run levels **2**, **3**, and **4** are available as normal operating states.

On your computer, the *run-levels* **0** and **1** are reserved states for shutting the system down, and *run-levels* **2**, **3**, and **4** are available as normal operating states.

If this is the first time since power up that *init* has entered a run level other than single-user state, *init* first scans **/etc/inittab** for **boot** and **bootwait** entries [see *inittab*(4)]. These entries are performed before any other processing of **/etc/inittab** takes place, providing that the run level entered matches that of the entry. 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. *init* then scans **/etc/inittab** and executes all other entries that are to be processed for that run level.

In a multiuser environment, **/etc/inittab** is set up so that *init* will create a *getty* process for each terminal that the administrator sets up to respawn.

To spawn each process in **/etc/inittab**, *init* reads each entry and for each entry that should be respawned, it forks a child process. After it has spawned all of the processes specified by **/etc/inittab**, *init* waits for one of its descendant processes to die, a powerfail signal, or a signal from another *init* or *telinit* process to change the system's run level. When one of these conditions occurs, *init* re-examines **/etc/inittab**. New entries can be added to **/etc/inittab** at any time; however, *init* still waits for one of the above three conditions to occur before re-examining **/etc/inittab**. To get around this, an **init Q** or **init q** command wakes *init* to re-examine **/etc/inittab** immediately.

When *init* comes up at boot time and whenever the system changes from the single-user state to another run state, *init* sets the *ioctl*(2) states of the virtual console to those modes saved in the file **/etc/ioctl.syscon**. This file is written by *init* whenever the single-user state is entered.

When a run level change request is made, *init* sends the warning signal (**SIGTERM**) to all processes that are undefined in the target run level. *init* waits 5 seconds before forcibly terminating these processes via the kill signal (**SIGKILL**).

The shell running on each terminal will terminate when the user types an end-of-file or hangs up. When *init* receives a 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 *init* receives a *powerfail* signal (**SIGPWR**) it scans **/etc/inittab** for special entries of the type *powerfail* and *powerwait*. 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 during the powerdown of the operating system. Note that in the single-user states, **S** and **s**, only *powerfail* and *powerwait* entries are executed.

telinit

telinit, which is linked to **/etc/init**, is used to direct the actions of *init*. It takes a one-character argument and signals *init* to take the appropriate action.

FILES

/etc/inittab
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/dev/console
/dev/contty

SEE ALSO

getty(1M), login(1), sh(1), shutdown(1M), stty(1), who(1), sulogin(1M), termio(7).

kill(2), gettydefs(4), inittab(4), utmp(4), in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

DIAGNOSTICS

If *init* finds that it is 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 in the entry, and generate an error message on the system console. It will then refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user-spawned *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 **/etc/inittab**.

When attempting to boot the system, failure of *init* to prompt for a new run level may be because the virtual system console is linked to a device other than the physical system console.

WARNINGS

init and *telinit* can be run only by someone who is super-user.

The **S** or **s** state must not be used indiscriminately in the **/etc/inittab** file. A good rule to follow when modifying this file is to avoid adding this state to any line other than the **initdefault**.

The change to **/etc/gettydefs** described in the **WARNINGS** section of the *gettydefs*(4) manual page will permit terminals to pass 8 bits to the system as long as the system is in multiuser state (run level greater than 1). When the system changes to single-user state, the *getty* is killed and the terminal attributes are lost. To permit a terminal to pass 8 bits to the system in single-user state, after you are in single-user state, type:

stty −istrip cs8

The **/etc/TIMEZONE** file must exist.

NAME

inskern − install a kernel

SYNOPSIS

inskern [−r root_directory **] [−g**shutdown_grace_period **]** kernel_file

DESCRIPTION

The *inskern* command installs a new kernel. By default, the configuration directory is **/etc/conf**; this may be overridden by setting the environment variable **$ROOT** (see *insdriver*(1)) or by using the **−r** option on the command line. This root will be referred to as **$ROOT** throughout this manual entry.

Installing a new kernel consists of linking the new kernel to **/unix**, shutting down the system, and booting the new kernel.

The *inskern* command does the following. It links, if possible, and otherwise copies **$ROOT/etc/conf/kconfig.d/***kernel_file***.d/***kernel_file* to */kernel_file*, copies configuration files from **/etc/conf/cf.d** to **/etc/conf/cf.d/OLD** and from **$ROOT/etc/conf/kconfig.d/***kernel_file***.d** to **/etc/conf/cf.d**, echoes *kernel_file* to **/etc/.new_unix**, and executes **/etc/shutdown**. **/etc/shutdown** moves the current kernel (**/unix**) to **/OLD.unix** and links */kernel_file* to **/unix**. Upon reboot, the default, **/unix**, will be the new kernel. If the new kernel cannot be linked to **/unix**, **/OLD.unix** is moved back to **/unix** and the configuration files are moved back from **/etc/conf/cf.d/OLD** to **/etc/conf/cf.d**.

Should the new kernel boot fail, reboot, and when the message:

    Booting the UNIX System...

is displayed, quickly press the space bar and type **/OLD.unix** when prompted for the name of the kernel. After the system is up, move **/OLD.unix** to **/unix**, and move **/etc/conf/cf.d/OLD/*** to **/etc/conf/cf.d**.

EXAMPLES

The following command line reboots the system with **/tmp/etc/conf/kconfig.d/unix.3.d/unix.3** as the new kernel:

    inskern −r /tmp unix.3

The following command reboots the system with **/etc/conf/kconfig.d/unix.3.d/unix.3** as the new kernel:

    inskern unix.3

FILES

/etc/.new_unix

SEE ALSO

kconfig(1), shutdown(1M).
insdriver(1) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

WARNINGS

The command *inskern* must be executed by the **root** user.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

install − install commands

SYNOPSIS

/etc/install [**−c** dira] [**−f** dirb] [**−i**] [**−n** dirc] [**−m** mode] [**−u** user] [**−g** group] [**−o**] [**−s**] file [dirx ...]

DESCRIPTION

The *install* command is 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 except **−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 except **−c** and **−f**.

−m *mode* — The mode of the new file is set to *mode*. Only available to the superuser.

−u *user* The owner of the new file is set to *user*. Only available to the superuser.

−g *group* The group id of the new file is set to *group*. Only available to the superuser.

−o If *file* is found, this option saves the "found" file by copying it to **OLD***file* in the directory in which it was found. This option is useful when installing a frequently used file such as */bin/sh* or */etc/getty*, where the existing file cannot be removed. May be used alone or with any other options except **−c**.

−s Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

make(1) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

installpkg — install package

SYNOPSIS

installpkg

DESCRIPTION

The *installpkg* command is used to install a UNIX system software package.

You will have to be *root* to install certain packages successfully.

You will be prompted to insert the floppy disk that the installation package resides on. Everything else is automatic.

LIMITATIONS

You must invoke *installpkg* on the console.

SEE ALSO

displaypkg(1), removepkg(1).

NAME

ipcrm – remove a message queue, semaphore set, or shared memory id

SYNOPSIS

ipcrm [options]

DESCRIPTION

The *ipcrm* command 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 *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

ipcs – report inter-process communication facilities status

SYNOPSIS

ipcs [options]

DESCRIPTION

ipcs prints certain information about active inter-process 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*:

–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 subject to these options:

–a Use all print *options*. (This is a shorthand notation for **–b**, **–c**, **–o**, **–p**, and **–t**.)

–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.

–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).

–X Print information about XENIX interprocess communication, in addition to the standard interprocess communication status. The XENIX process information describes a second set of semaphores and shared memory. Note that the **-p** option does not print process number information for XENIX shared memory, and the **-t** option does not print time information about XENIX semaphores and shared memory.

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 facility; they do *not* determine which facilities will be listed.

T (all) Type of the facility:

- **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 *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** 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:

- **R** if a process is waiting on a *msgrcv*.
- **S** if a process is waiting on a *msgsnd*.
- **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 set refers to permissions of others in the user-group of the facility entry. The last set refers 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. Permissions are indicated as follows:

- **r** if read permission is granted
- **w** if write permission is granted
- **a** if alter permission is granted
- **–** if the indicated permission is *not* 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.
LSPID	(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 namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

msgop(2), semop(2), shmop(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

WARNING

If the user specifies either the −C or −N flag, the real and effective UID/GID will be set to the real UID/GID of the user invoking *ipcs.*

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

NAME

ismpx — return windowing terminal state

SYNOPSIS

ismpx [—s]

DESCRIPTION

The *ismpx* command reports whether its standard input is connected to a multiplexed *xt*(7) channel; i.e., whether it's running under *layers*(1) or not. It is useful for shell scripts that download programs to a windowing terminal or depend on screen size.

The *ismpx* command prints **yes** and returns **0** if invoked under *layers*(1), and prints **no** and returns **1** otherwise.

—s Do not print anything; just return the proper exit status.

EXIT STATUS

Returns **0** if invoked under *layers*(1), **1** if not.

SEE ALSO

jwin(1), layers(1), xt(7).

EXAMPLE

```
if ismpx —s
then
        jwin
fi
```

NAME

join – relational data base operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

The *join* command 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 [see *sort*(1)].

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:

–a*n* 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*.

–j*n 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.

–t*c* 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 –j1 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

jterm – reset layer of windowing terminal

SYNOPSIS

jterm

DESCRIPTION

The *jterm* command is used to reset a layer of a windowing terminal after downloading a terminal program that changes the terminal attributes of the layer. It is useful only under *layers*(1). In practice, it is most commonly used to restart the default terminal emulator after using an alternate one provided with a terminal-specific application package. For example, on the AT&T TELETYPE 5620 DMD terminal, after executing the *hp2621*(1) command in a layer, issuing the *jterm* command will restart the default terminal emulator in that layer.

EXIT STATUS

Returns **0** upon successful completion, **1** otherwise.

NOTE

The layer that is reset is the one attached to standard error; that is, the window you are in when you type the *jterm* command.

SEE ALSO

layers(1).

NAME

jwin – print size of layer

SYNOPSIS

jwin

DESCRIPTION

The *jwin* command runs only under *layers*(1) and is used to determine the size of the layer associated with the current process. It prints the width and the height of the layer in bytes (number of characters across and number of lines, respectively). For bit-mapped terminals only, it also prints the width and height of the layer in bits.

EXIT STATUS

Returns **0** on successful completion, **1** otherwise.

DIAGNOSTICS

If *layers*(1) has not been invoked, an error message is printed:

jwin: not mpx

NOTE

The layer whose size is printed is the one attached to standard input; that is, the window you are in when you type the *jwin* command.

SEE ALSO

layers(1).

EXAMPLE

```
jwin
bytes:  86 25
bits:   780 406
```

NAME

kconfig − configure, build, and install a kernel

SYNOPSIS

kconfig [**−r** root_directory]

DESCRIPTION

The *kconfig* command provides a menu interface to configure, build, or install a kernel. For additional information, see the section "Using *kconfig* to Tailor Your System Kernel" in the "INTERACTIVE UNIX Operating System Maintenance Procedures" in the *INTERACTIVE UNIX Operating System Guide*.

By default, the root of the directory tree in which the configuration takes place, **etc/conf**, is /; this may be overridden by setting the environment variable **$ROOT** or by using the **−r** option on the command line. This root will be referred to as **$ROOT** throughout this manual entry.

System files are contained in **$ROOT/etc/conf/cf.d**. The *kconfig* command performs all modifications to the system files via menu choices.

The possible responses to *kconfig* menus are:

> a number corresponding to the menu item choice
> an **m**, to return to the previous menu
> a **q**, to quit and exit the program

Each of the above must be followed by **ENTER**.

Where a *kconfig* query ends with text within parentheses, that text is the default. Using **ENTER** will select the default. Otherwise, enter the requested information, followed by **ENTER**.

Menus

If the **$ROOT** directory has not been specified, *kconfig* will query for the needed information before the first menu appears.

Top Level Menu

```
MAIN MENU

 1) CONFIGURE KERNEL
 2) BUILD A KERNEL
 3) INSTALL A KERNEL

Enter Choice [1-3,q]:
```

Choice **1** will cause the Kernel Configuration Menu to be displayed (see Kernel Configuration Menu below).

Choice **2** will build the kernel by executing **$ROOT/etc/conf/bin/idbuild**. The kernel will be built as configured in choice **1**. If the kernel build is successful, *kconfig* will query for kernel installation. If the kernel is to be installed, **/etc/inskern** will be executed (see *inskern*(1)).

Choice **3** will put up the Install Kernel Menu, which is used to install a previously built kernel (see Install Kernel Menu below).

Install Kernel Menu

Before displaying the menu, *kconfig* prints a notice that a system shutdown is required to install a kernel (see *shutdown*(1M)). This menu displays all kernels contained in the **$ROOT/etc/conf/kconfig.d** directory. For example:

```
CHOOSE THE KERNEL TO INSTALL

  1) unix.1
  2) unix.2

Enter Choice [1-2,m,q]:
```

The number chosen indicates which kernel will be installed. After verifying the choice, *kconfig* queries for the shutdown grace period and executes **/etc/inskern** to install the kernel.

Kernel Configuration Menu

The kernel configuration menu presents menu choices for modifying system file(s). After returning from the Configuration Menu, if modifications were made, *kconfig* asks whether to save the modified system file(s).

```
CONFIGURATION MENU

  1) ADD DRIVER
  2) REMOVE DRIVER
  3) ADD FACILITY
  4) REMOVE FACILITY
  5) ADD DEFAULT PARAMETERS FOR MEMORY SIZE
  6) ADD TUNABLE PARAMETERS
  7) DISPLAY HIGH PERFORMANCE DISK DRIVER CONFIGURATION
  8) CONFIGURE HIGH PERFORMANCE DISK DRIVER

Enter Choice [1-8,m,q]:
```

Choice **1** puts up the Add Driver Menu, which is used to include kernel device driver modules in the set of configured modules (see Add Driver Menu below).

Choice **2** puts up the Remove Driver Menu, which is used to remove kernel device driver modules from the set of configured modules (see Remove Driver Menu below).

Choice **3** puts up the Add Facility Menu, which is used to include groups of kernel modules in the set of configured modules (see Add Facility Menu below).

Choice **4** puts up the Remove Facility Menu, which is used to remove groups of kernel modules from the set of configured modules (see Remove Facility Menu below).

Choice **5** puts up the Add Default Parameters For Memory Size Menu. This menu is used to set predefined tunable parameters based on system memory size (see Add Default Parameters Menu below).

Choice **6** adds user-entered tunable parameters to the **stune** system file. *kconfig* queries for the parameter name and value. Using **RETURN** terminates the additions. If this is a new parameter (one that is not already in the system file **$ROOT/etc/conf/cf.d/mtune**), *kconfig* will query for minimum, maximum, and default values.

Choice **7** displays the current configuration of the High Performance Disk Driver if it has previously been configured using choice **8** below.

Choice **8** presents a series of inquiries and menus used to configure the High Performance Disk Driver (see "Configure High Performance Disk Driver" below).

Add Driver Menu

This menu lists known drivers that are currently able to be configured in the **sdevice** system file. The menu items are examples only and will vary according to the **sdevice** file contents.

```
CHOOSE A DRIVER TO ADD TO THE CURRENT CONFIGURATION

 1) Bell Technologies Hub Board
 2) Bell Tech ICC Board Driver
 3) Built-in Mouse Driver
 4) Line Printer Driver
 5) Logitech Bus Mouse Driver
 6) Microsoft Bus Mouse Driver
 7) Wangtek Cartridge Tape Driver

 Enter Choice [1-7,m,q]:
```

After verifying the choice, *kconfig* includes the corresponding kernel driver module in the **sdevice** system file. It does this by setting the second field to **Y** for that entry.

Remove Driver Menu

This menu lists drivers that can be removed from the **sdevice** system file. These drivers are modules currently included in the **sdevice** file (entries having the second field set to **Y**). The menu items are examples only and will vary according to the **sdevice** file contents.

```
CHOOSE A DRIVER TO REMOVE FROM THE CURRENT CONFIGURATION

 1) Serial I/O Driver
 2) Floppy Disk Driver
 3) Shell Layers Driver

 Enter Choice [1-3,m,q]:
```

After verifying the choice, *kconfig* removes the corresponding kernel driver module from the system file by setting the second field to **N** for that entry.

Add Facility Menu

This menu lists kernel facilities whose modules are not in the set of currently configured modules. The menu items are examples only and will vary according to the contents of the source file.

```
CHOOSE A FACILITY TO ADD TO THE CURRENT CONFIGURATION

  1) Kernel Debugger
  2) MS-DOS File System Service
  3) Unix Kernel Profiler
  4) SunRiver Fiber Optic Station
  5) Operating System Messages
  6) STREAMS Facilities
  7) 2 Kilobyte File System
  8) Host Based TCP/IP
  9) Network File System

  Enter Choice [1-9,m,q]:
```

After verifying the choice, *kconfig* adds the corresponding group of kernel modules to the **sdevice** file by setting the second field to **Y** for that entry.

Remove Facility Menu

This menu lists kernel facilities whose modules are included in the current **sdevice** file. The menu items are examples only and will vary according to the system file contents.

```
CHOOSE A FACILITY TO REMOVE FROM THE CURRENT CONFIGURATION

  1) MS-DOS File System Service
  2) Inter Process Communication
  3) Shared Memory
  4) Weitek 1167 Support
  5) XENIX Shared Memory
  6) XENIX Semaphores
  7) XENIX File System

  Enter Choice [1-7,m,q]:
```

After verifying the choice, *kconfig* removes the corresponding group of kernel modules from the system file.

Add Default Parameters For Memory Size Menu

The standard kernel supplied with the system is optimized for a system with 4 MB of RAM. Although all the memory that is located when the system is booted will be used, system performance will increase if the kernel is tuned to make more efficient use of different amounts of system memory. Using this option, for higher amounts of RAM, more memory will be dedicated to system buffers and other kernel structures as well as increasing certain process-related parameters. If the amount of memory you have installed falls between the available choices, choose the next lowest option. The INTERACTIVE UNIX System will operate unreliably if the system does not have as much memory as the kernel expects.

This menu lists the memory size categories for which there are available predefined tunable parameter sets:

```
     CHOOSE THE CLOSEST MEMORY SIZE

1)   4MB
2)   8MB
3)   16MB

Enter choice [1-3,m,q]:
```

Type the number corresponding to the amount of RAM installed on your system. After you have selected an option, the system will attempt to determine whether you are increasing or decreasing your memory size from that previously configured and will prompt for confirmation. When you are increasing your memory size, if an existing value for a parameter is larger than the value for that parameter in the defaults for the chosen memory size, the parameter will retain its present higher value. However, if you are decreasing your memory size, all of the default values will be used and a warning will be given stating that reducing tunable parameters could cause the system to run out of resources under certain circumstances, and to use the Add Tunable Parameters submenu of the Configuration Menu to increase the resources being exceeded if this happens.

Configure High Performance Disk Driver

The High Performance Disk Driver (HPDD) is a system of device and controller drivers that together provide fast, consistent support for many disk and tape devices. The devices that are under the domain of the HPDD are fixed disk controllers and SCSI tape drives (see the INTERACTIVE UNIX Operating System Release Notes for a list of supported devices). In addition, the HPDD supports a RAM disk, which is created by reserving a portion of the computer's available memory, which is then treated as if it were a disk storage device (see sections 6 and 7 in the "INTERACTIVE UNIX System Maintenance Procedures" in the *INTERACTIVE UNIX Operating System Guide* for more information about RAM disks). To support a configuration of these devices that differs from the default configuration documented in the "INTERACTIVE UNIX System Installation Instructions" in the *INTERACTIVE UNIX Operating System Guide*, you must reconfigure the HPDD. Even if your configuration does not require you to reconfigure the HPDD, you can improve performance at boot time by doing so. Under the default configuration, upon booting, the HPDD must determine the type of fixed-disk controller present; when the HPDD has been reconfigured, this information is known, reducing the startup time.

Use this option to:

> Add, remove, change, or specify the type of fixed disk controller.
>
> Change the interrupt vector used by a SCSI host adapter.
>
> Add, remove, or change the type of tape drive connected to a SCSI host adapter.
>
> Add, remove, or change the size of a RAM disk.

Once you have reconfigured the HPDD, if you subsequently make any changes to your system with respect to these devices (other than replacing a standard AT controller of one type with a standard AT controller of a different type or adding or removing fixed disk drives), you will need to reconfigure it again.

For complete information on configuring the HPDD, refer to the section "Hardware Compatibility and Configuration" in the "INTERACTIVE UNIX System Maintenance Procedures."

Description File

The kernel module descriptions and group designations used by the configuration menus to add or remove drivers and facilities are obtained from the file **$ROOT/etc/conf/kconfig.d/description**. This file lists the kernel module names and description and the group designations.

Description file format

module name	–	–	group name	–	description

Example

asy	–	–	io	–	AT serial I/O Driver
hub	–	–	io	–	AT Bell Technologies Hub board
lp	–	–	io	–	AT Line Printer Driver
sxt	–	–	io	–	Shell Layers Driver
wt	–	–	io	–	AT Wangtek cartridge tape driver
–	–	–	ipc	–	Inter Process Communication
ipc	–	–	ipc	–	ipc common routines
msg	–	–	ipc	–	ipc message facility
sem	–	–	ipc	–	ipc semaphore facility

The module name is the name of the system file entry. The group name is the group for that module; group name *io* designates I/O drivers, which are listed on the Add or Remove Drivers Menus. A "–" in the module name designates the description used on the Add or Remove Facilities Menus for the designated group.

For example, the group name *ipc* designates a kernel facility. The description field on the line with module name "–" and group name *ipc* is the description used on the Add or Remove Facilities Menus. The lines with module names and group *ipc* designate the module list for that group. In this case, adding or removing the facility *Inter Process Communication* would add or remove the modules *ipc*, *msg*, and *sem*.

Note that the "–" columns in the above example must be present. These fields are reserved for future function information.

FILES

$ROOT/etc/conf/kconfig.d

SEE ALSO

idbuild(1M), inskern(1), shutdown(1M).
"INTERACTIVE UNIX Operating System Maintenance Procedures" in the *INTERACTIVE UNIX Operating System Guide*.

WARNINGS

The *kconfig* command can be executed only by user **root**. To install a kernel, *kconfig* should be executed from the / (**root**) directory.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

kill — terminate a process

SYNOPSIS

kill [−signo] PID ...

DESCRIPTION

The *kill* command 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 *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

killall – kill all active processes

SYNOPSIS

/etc/**killall** [signal]

DESCRIPTION

The *killall* command is used by **/etc/shutdown** to kill all active processes not directly related to the shutdown procedure.

The *killall* command terminates all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

The *killall* command sends *signal* [see *kill*(1)] to all 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) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

WARNINGS

The *killall* command can be run only by the super-user.

NAME

labelit – provide labels for file systems

SYNOPSIS

/etc/labelit special [fsname volume [**–n**]]

DESCRIPTION

The *labelit* command can be used to provide labels for unmounted disk file systems or file systems being copied to tape. The **–n** option provides for initial labeling only. (This destroys previous contents.)

With the optional arguments omitted, *labelit* prints current label values.

The *special* name should be the physical disk section (e.g., **/dev/dsk/0s3**). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (e.g., **root**, **u1**, etc.) of the file system.

Volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette, or tape.

For file systems on disk, *fsname* and *volume* are recorded in the super block.

SEE ALSO

sh(1).
fs(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

layers – layer multiplexer for windowing terminals

SYNOPSIS

layers [–s] [–t] [–d] [–p] [–f file] [layersys-prgm]

DESCRIPTION

The *layers* command manages asynchronous windows [see *layers*(5)] on a windowing terminal. Upon invocation, *layers* finds an unused *xt*(7) channel group and associates it with the terminal line on its standard output. It then waits for commands from the terminal.

Command-line options:

–s Reports protocol statistics on standard error at the end of the session after you exit from *layers*. The statistics may be printed during a session by invoking the program *xts*(1M).

–t Turns on *xt*(7) driver packet tracing, and produces a trace dump on standard error at the end of the session after you exit from *layers*. The trace dump may be printed during a session by invoking the program *xtt*(1M).

–d If a firmware patch has been downloaded, prints out the sizes of the text, data, and bss portions of the firmware patch on standard error.

–p If a firmware patch has been downloaded, prints the downloading protocol statistics and a trace on standard error.

–f *file* Starts *layers* with an initial configuration specified by *file*. Each line of the file represents a layer to be created, and has the following format:

origin_x origin_y corner_x corner_y command_list

The coordinates specify the size and position of the layer on the screen in the terminal's coordinate system. If all four are *0*, the user must define the layer interactively. *command_list*, a list of one or more commands, must be provided. It is executed in the newlayer using the user's shell (by executing: **$SHELL –i –c** "*command_list*"). This means that the last command should invoke a shell, such as **/bin/sh**. (If the last command is not a shell, then, when the last command has completed, the layer will not be functional.)

layersys-prgm
A file containing a firmware patch that the *layers* command downloads to the terminal before layers are created and *command_list* is executed.

Each layer is in most ways functionally identical to a separate terminal. Characters typed on the keyboard are sent to the standard input of the UNIX system process attached to the current layer (called the host process), and characters written on the standard output by the host process appear in that layer. When a layer is created, a separate shell is established and bound to the layer. If the environment variable **SHELL** is set, the user will get that shell: otherwise, **/bin/sh** will be used. In order to enable communications with other users via *write*(1), *layers* invokes the command *relogin*(1M) when the first layer is created. *relogin*(1M) will reassign that layer as the user's logged-in terminal. An alternative layer can be designated by using *relogin*(1M) directly. *layers* will restore the original assignment on termination.

Layers are created, deleted, reshaped, and otherwise manipulated in a terminal-dependent manner. For instance, the AT&T TELETYPE 5620 DMD terminal provides a mouse-activated pop-up menu of layer operations. The method of ending a *layers* session is also defined by the terminal.

EXAMPLE

layers −f startup

where **startup** contains

8 8 700 200 date ; pwd ; exec $SHELL
8 300 780 850 exec $SHELL

NOTES

The *xt*(7) driver supports an alternate data transmission scheme known as ENCODING MODE. This mode makes *layers* operation possible even over data links which intercept control characters or do not transmit 8-bit characters. ENCODING MODE is selected either by setting a configuration option on your windowing terminal or by setting the environment variable **DMDLOAD** to the value *hex* before running *layers*:

    export DMDLOAD; DMDLOAD=hex

If, after executing **layers -f** *file*, the terminal does not respond in one or more of the layers, often the last command in the *command-list* for that layer did not invoke a shell.

WARNING

When invoking *layers* with the **−s**, **−t**, **−d**, or **−p** options, it is best to redirect standard error to another file to save the statistics and tracing output (e.g., **layers −s 2>stats**); otherwise all or some of the output may be lost.

FILES

/dev/xt??[0-7]
/usr/lib/layersys/lsys.8;7;3
/usr/lib/layersys/lsys.8;?;?

SEE ALSO

relogin(1M), sh(1), write(1), wtinit(1M), xts(1M), xtt(1M), xt(7).

libwindows(3X), layers(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

lef, dtou, utod – DOS-to-UNIX and UNIX-to-DOS line ending filters

SYNOPSIS

lef [**–z**] [input_file] [output_file]
dtou [DOS_input_file] [UNIX_output_file]
utod [**–z**] [UNIX_input_file] [DOS_output_file]

DESCRIPTION

lef is a filter that converts MS-DOS (DOS) format input to UNIX System format and UNIX System format input to DOS format. It automatically determines the format of the input and converts it to the opposite format. The *dtou* filter converts from DOS to UNIX System format, and *utod* converts from UNIX to DOS format.

When going from DOS to UNIX, carriage return/line-feed is converted into a new-line character, and if there is a **CTRL z** at the end of the file, it is removed. When going from UNIX to DOS, a new-line character is converted into carriage return/line-feed. The **–z** option will cause a **CTRL z** to be written at the end of the DOS output for backward compatibility with older DOS programs that use **CTRL z** to determine end-of-file.

If no arguments are specified, these programs read from standard input and write to standard output. If one argument is specified, that is used as the input file. If two arguments are specified, the first is the input file and the second is the output file.

To prevent accidental conversion of binary files, if there are certain control characters or any 8-bit characters in the input, or if the filter cannot otherwise determine that the input is either in DOS or UNIX System format, *lef* will write a "No translation" message to standard error, and the entire input will be copied to the output without being converted. To force translation, use either *dtou* or *utod*, which will do the specified conversion without qualification.

Examples

To convert a file from its original format to the opposite format:

```
lef infile outfile
```

To convert a DOS format file to UNIX System format:

```
dtou dosfile unixfile
```

To convert a UNIX System format file to DOS format, appending a **CTRL z** to the end:

```
utod –z unixfile dosfile
```

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

line – read one line

SYNOPSIS

line

DESCRIPTION

The *line* command copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on **EOF** and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

SEE ALSO

sh(1).
read(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

link, unlink — link and unlink files and directories

SYNOPSIS

/etc/link file1 file2
/etc/unlink file

DESCRIPTION

The *link* command is used to create a file name that points to another file. Linked files and directories can be removed by the *unlink* command; however, it is strongly recommended that the *rm*(1) and *rmdir*(1) commands be used instead of the *unlink* command.

The only difference between *ln*(1) and *link/unlink* is that the latter do exactly what they are told to do, abandoning all error checking. This is because they directly invoke the *link*(2) and *unlink*(2) system calls.

SEE ALSO

rm(1).
link(2), unlink(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

WARNINGS

These commands can be run only by the super-user.

NAME

lmouse – use a LOGITECH 2 button serial mouse as input device

SYNOPSIS

lmouse [devname]

DESCRIPTION

The *lmouse* utility allows a LOGITECH 2 button serial mouse to be used as a keyboard input device when hooked up to a serial port on the system. Moving the mouse generates the same scan codes generated when the user presses the arrow keys on the keyboard. Thus the mouse can be used to position the cursor in most applications that respond to the arrow keys. In addition, pressing the left button on the mouse simulates the **ENTER** key, and pressing the right button simulates the **ESC** key.

Argument

devname is a string describing the full path name to the serial device to which the mouse is attached. If no argument is specified, *lmouse* defaults to **/dev/tty00**. If *devname* is specified, *lmouse* tries to open the device for reading and writing.

Example

```
lmouse /dev/tty01 &
```

Run *lmouse* in the background with a LOGITECH 2 button serial mouse hooked up to the second serial port on the system.

Notes

It is recommended that *lmouse* be invoked in the background or as part of an initialization script when the system starts up. The scan codes generated by *lmouse* cannot be modified by the user.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

loadfont – list or change font information in the RAM of the video card

SYNOPSIS

loadfont

loadfont −f filename

loadfont codepage

loadfont −l

loadfont −d

loadfont −m mode

DESCRIPTION

The *loadfont* utility allows a user to load and activate a different font into the RAM of the video card used by the console of the INTERACTIVE UNIX Operating System. It can also be used to display information about the font currently in use. In addition, the **−m** option can be used to change the size of the characters on the screen; it can also be used to change the number of lines or colors, e.g., to run an application at the console at 43 lines at a time instead of 25. *loadfont* will always read from standard output; this will allow a system administrator to use it from a remote terminal.

Options

loadfont

When used without arguments, *loadfont* displays the different ways the command can be used, as shown in the synopsis.

loadfont **−f** *filename*

This command reads the contents of *filename* and subsequently loads the font specified in the file into the RAM of the video card. If the file does not have the correct format, an error message is produced.

loadfont codepage

If *codepage* is the name of a hard-coded font available for the current font size, this font will be loaded into the RAM of the video card and activated. Available font names are listed when the **−l** option is used. If the *codepage* argument specified is not the name of a valid font, an error message will be produced.

loadfont **−l**

This option displays a short description of the fonts that are hard-coded into the program and the name that can be passed as a *codepage*. Only the fonts that match the current font size are listed. *loadfont* **−l** also displays the different character modes supported by *loadfont* and the exact name that should be used with the **−m** option. Here is a sample output:

Codepages supported for this size font are:

Name	Description
437	IBM 437 codepage
8859	ISO 8859-1 codeset
8859g	ISO 8859-1 with graphics
850	IBM 850 codepage

Different possible text modes supported are:

Name	Description
E80x43	EGA 80 columns 43 lines
E40x25	EGA 40 columns 25 lines
E80x25	EGA 80 columns 25 lines
V40x25	VGA 40 columns 25 lines
V80x25	VGA 80 columns 25 lines

8859g means the 8859-1 codeset with box-drawing characters in column 9 of the table (characters 0x90 to 0x9a).

loadfont **−d**

This *read*s the font information from the video RAM and *write*s it to standard output in a format compatible with the Binary Distribution Format version 2.1 as developed by Adobe Systems, Inc.

loadfont **−m** *mode*

This will attempt to change the mode of the console as specified. This will result in having a different font size and/or different number of lines and columns on the screen. The *mode* that can be specified should be one of the choices listed above in the *loadfont* **−l** output. If an invalid argument is specified, an error message is produced.

Fonts

A font is the representation of characters by images. The need to use different fonts can be imposed by:

1. The codeset used to represent the characters internally.
2. The resolution used to display the characters.

Each font contains exactly 256 images. All fonts supported are fixed size (constant width and constant height), i.e., each character takes the same amount of space on the screen. When the monitor is not being used in graphics mode, the *loadfont* utility allows a user to modify the font used by the video card, so different images are displayed on the screen of the console for the various characters. Depending on the type of video card used, different text modes can be supported by the same video card. They typically differ by the number of pixels used to represent a single character. For each character, the same number of pixels is used. For the standard video cards, the different resolutions supported (all or a subset) are:

8 by 8 (8 horizontally and 8 vertically)

8 by 14

8 by 16

When *loadfont* is invoked to modify the existing font, it will attempt to do so for the font size currently in use. Use the **−m** option to switch to another font size.

loadfont **and** *ttymap*

There is an almost one-to-one relationship between the use of the *loadfont* utility and the *ttymap* utility. Whereas *loadfont* is used to list or modify the images that correspond with the various characters, the *ttymap* utility is used to determine how characters are generated from the keyboard and which code (a single byte code) will be used to represent the character internally. The default representation is the IBM extended ASCII codeset, often also referred to as "IBM codepage 437." A *ttymap* sample input file is supplied that can be used for this codeset on a console with a U.S. keyboard (**usa.map**). When a different keyboard is used, a different *ttymap* input file is required (e.g., **french.map** for a French keyboard).

When a different codeset is used, both a different *ttymap* input file and a different font are required. For the most popular *codesets*, fonts are hard-coded into the *loadfont* program for the 8 by 16 resolution (see "Fonts"). If these fonts do not satisfy your needs (because you want to use a different font size or because a customized font is required, e.g., a Greek font), a *loadfont* description file to be used with the **−f** option is needed. A sample file that describes the IBM extended ASCII font for an 8 by 16 resolution is supplied (**vga437.bdf**). A second sample file, **646g.bdf**, contains a font file for German ASCII. See *ttymap*(1) and *loadfont*(5) for additional details.

WARNING

When an attempt is made to switch to a mode that the video card does not support (e.g., a switch to EGA on a VGA card that has no EGA mode) you will get a blank screen. There is nothing wrong with the system; simply type in the command to set the mode back, e.g.:

```
loadfont −m V80x25
```

FILES

/usr/lib/loadfont/vga437.bdf	sample Bitmap Distribution Format (BDF) file for IBM 437 font on a VGA
/usr/lib/loadfont/646g.bdf	sample BDF file for German ASCII

SEE ALSO

ttymap(1), display(7).
loadfont(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

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."

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.

If you make a mistake in the login procedure you will receive the message:

    Login incorrect

and a new login prompt will appear. If you make five incorrect login attempts, all five may be logged in **/usr/adm/loginlog** (if it exists) and the line will be dropped.

If you do not complete the login successfully within a certain period of time (e.g., 1 minute), you will probably be silently disconnected.

After a successful login, the user ID, the group IDs, the working directory, and the command interpreter (usually *sh*(1) or *csh*(1)) are initialized. If the shell **/bin/sh** is running, accounting files are updated, the procedure **/etc/profile** is performed, the message-of-the-day (if any) is printed, and the file **.profile** in the working directory is executed, if it exists. If the shell **/bin/csh** is running, the **.login** and **.cshrc** files in the working directory are executed, if they exist. 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 the named directory becomes the root directory, the starting point for path searches for path names beginning with a slash (/). 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* 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:

L*n*=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 exceptions. The variables **HOME, PATH, SHELL, MAIL, IFS, TZ, HZ, CDPATH,** and **LOGNAME** 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

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/*your-name*	mailbox for user *your-name*
/usr/adm/loginlog	record of failed login attempts
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system profile (**/bin/sh** only)
.profile	user's login profile (**/bin/sh** only)

SEE ALSO

mail(1), sh(1), newgrp(1M), su(1M).
loginlog(4), passwd(4), profile(4), environ(5) in the *INTERACTIVE SDS Guide and Programmer's Reference 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. If you attempted to execute *login* as a command without using the shell's *exec* internal command or attempted to execute *login* as a command from a shell other than the initial shell, you must execute *login* from the lowest level shell.

NOTES

The file **/etc/default/login** contains special login information including the flag PASSREQ. When PASSREQ is set to YES (PASSREQ=YES), users will be required to have a password. When a user without a password logs in and PASSREQ=YES, the user will be forced to add a password before the user is allowed access to the system. One exception to this requirement is if password aging is turned on for the user and the NULL password has not been aged. In this case, the user will be allowed to access the system without a password until the NULL password has been aged, or until the **root** user forces the password to be aged (*passwd* **−f** command).

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

logname − get login name

SYNOPSIS

logname

DESCRIPTION

The *logname* command returns the contents of the environment variable **$LOGNAME**, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1).
logname(3X), environ(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

lp, cancel – send/cancel requests to an LP print service

SYNOPSIS

lp [*printing options*] *files*

lp **–i** *id printing options*

cancel [*ids*] [*printers*]

DESCRIPTION

The first form of the *lp* shell command arranges for the named files and associated information (collectively called a *request*) to be printed. If no file names are specified on the shell command line, the standard input is assumed. The standard input may be specified along with named *files* on the shell command line using the file name. The *files* will be printed in the order they appear on the shell command line.

The second form of *lp* is used to change the options for a request. The print request identified by the *request–id* is changed according to the printing options specified with this shell command. The printing options available are the same as those with the first form of the *lp* shell command. If *request–id* has finished printing, the change is rejected. If the *request–id* is already printing, it will be stopped and restarted from the beginning, unless the **–P** option has been given.

lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel, change, or find the status of the request. (See the section on *cancel* for details about canceling a request, the previous paragraph for an explanation of how to change a request, and *lpstat*(1) for information about checking the status of a print request.)

Sending a Print Request

The first form of the *lp* command is used to send a print request to a particular printer or group of printers.

Options to *lp* must always precede file names but may be listed in any order. The following options are available for *lp*:

–c Makes copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will not be copied. 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.

–d *dest* Prints this request using *dest* as the printer or class of printers. Under certain conditions (lack of printer availability, capabilities of printers, and so on), 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)].

−f *form-name* [**−d any**]

Prints the request on the form *form-name*. The LP print service ensures that the form is mounted on the printer. If *form-name* is requested with a printer destination that cannot support the form, the request is rejected. If *form-name* has not been defined for the system or if the user is not allowed to use the form, the request is rejected [see *lpforms*(1M)]. When the **−d any** option is given, the request is printed on any printer that has the requested form mounted and can handle any other needs of the print request.

−H *special-handling*

Prints the request according to the value of *special-handling*. Acceptable values for *special-handling* are **hold**, **resume**, and **immediate**, as defined below:

hold Won't print the request until notified. If already printing, stops it. Other print requests will go ahead of a held request until it is resumed.

resume Resumes a held request. If it had been printing when held, it will be the next request printed, unless subsequently bumped by an **immediate** request.

immediate

(Available only to LP administrators)
Prints the request next. If more than one request is assigned **immediate**, the requests are printed in the reverse order queued. If a request is currently printing on the desired printer, you have to put it on hold to allow the immediate request to print.

−m Sends mail [see *mail*(1)] after the files have been printed. By default, no mail is sent upon normal completion of the print request.

−n *number*

Prints *number* copies of the output. (Default is 1.)

−o *option*

Specifies printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **−o** keyletter more than once. The standard interface recognizes the following options:

nobanner

Does not print a banner page with this request. (The administrator can disallow this option at any time.)

nofilebreak

Does not insert a form feed between the files given if submitting a job to print more than one file.

length=*scaled-decimal-number*

Prints the output of this request with pages *scaled-decimal-number* lines long. A *scaled-decimal-number* is an optionally scaled decimal number that gives a size in lines, columns, inches, or centimeters, as appropriate. The scale is indicated by appending the letter "i" (for inches) or the letter "c" (for centimeters). For length or width settings, an unscaled number indicates lines or columns; for line pitch or character pitch settings, an unscaled number indicates lines per inch or characters per inch (the same as a number scaled with "i"). For example, **length=66** indicates a page length of 66 lines, **length=11i** indicates a page length of 11 inches, and **length=27.94c** indicates a page length of 27.94 centimeters.

This option cannot be used with the **−f** option.

width=*scaled-decimal-number*
Prints the output of this request with page-width set to *scaled-decimal-number* columns wide. (See the explanation above for *scaled-decimal-numbers.*) This option cannot be used with the **−f** option.

lpi=*scaled-decimal-number*
Prints this request for "lines per inch" with the line pitch set to *scaled-decimal-number* lines per inch. This option cannot be used with the **−f** option.

cpi=*scaled-decimal-number*
Prints this request for "characters per inch" with the character pitch set to *scaled-decimal-number* characters per inch. Character pitch can also be set to **pica** (representing 10 columns per inch) or **elite** (representing 12 columns per inch), or it can be **compressed,** which is as many columns as a printer can handle. There is no standard number of columns per inch for all printers; see the *terminfo*(4) database for the default character pitch for your printer. The **cpi** option cannot be used in conjunction with the **−f** option.

stty=*stty-option-list*
A list of options valid for the **stty** command. Enclose the list with quotes if it contains blanks.

−P *page-list*
Prints the page specified in *page-list*. This option can be used only if there is a filter available to handle it; otherwise, the print request will be rejected.

The *page-list* may consist of range(s) of numbers, single page numbers, or a combination of both. The pages will be printed in ascending order.

−q *priority-level*
Assigns this request *priority-level* in the printing queue. The values of *priority-level* range from 0, the highest priority, to 39, the lowest priority. If a priority is not specified, the default for the print service is used, as assigned by the system administrator.

−s Suppresses messages from *lp*(1) such as "request id is ...".

−S *character-set* **[−d any]**
−S *print-wheel* **[−d any]**

Prints this request using the specified *character-set* or *print-wheel*. If a form has been specified that requires a *character-set* or *print-wheel* other than the one specified with the **−S** option, the request is rejected.

For printers that take print wheels: if the *print-wheel* specified is not one listed by the administrator as acceptable for the printer involved in this request, the request is rejected unless the print wheel is already mounted on the printer. For printers that use selectable or programmable character sets: if the *character-set* specified is not one defined in the Terminfo database for the printer [see *terminfo*(4)] or is not an alias defined by the administrator, the request is rejected.

When the **−d any** option is used, the request is printed on any printer that has the print wheel mounted or any printer that can select the character set and can handle any other needs of the request.

−t *title* Prints *title* on the banner page of the output. The default is no title.

−T *content-type* **[−r]**

Prints the request on a printer that can support the specified *content-type*. If no printer accepts this type directly, a filter will be used to convert the content into an acceptable type. If the **−r** option is specified, a filter will not be used. If **−r** is specified but no printer accepts the *content-type* directly, the request is rejected. If the *content-type* is not acceptable to any printer, either directly or with a filter, the request is rejected.

−w Writes 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.

−y *mode-list*

Prints this request according to the printing modes listed in *mode-list*. The allowed values for *mode-list* are locally defined. This option can be used only if there is a filter available to handle it; if there is no filter, the print request will be rejected.

Canceling a Print Request

The *cancel* command cancels printer requests that were made by the *lp*(1) shell command. The shell command line arguments may be either *request-ids* [as returned by *lp*(1)] or *printer* names [for a complete list, use *lpstat*(1)]. Specifying a *request-id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request that 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.

NOTES

Printers for which requests are not being accepted will not be considered when the destination is **any**. (Use the **lpstat −a** command to see which printers are accepting requests.) On the other hand, if a request is destined for a class of printers and the class itself is accepting requests, *all* printers in the class will be considered, regardless of their acceptance status, as long as the printer class is accepting requests.

WARNING

For printers that take mountable print wheels or font cartridges, if you do not specify a particular print wheel or font with the **−S** option, whichever happens to be mounted at the time your request prints will be used. Use the **lpstat −p −l** command to see what print wheels are available. For printers that have selectable character sets, you will get the standard set if you don't give the **−S** option.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1), accept(1M), lpadmin(1M), lpfilter(1M), lpforms(1M), lpsched(1M), lpusers(1M).

terminfo(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

lpadmin – configure the LP print service

SYNOPSIS

/usr/lib/lpadmin –p printer options
/usr/lib/lpadmin –x dest
/usr/lib/lpadmin –d [dest]
/usr/lib/lpadmin –S print-wheel –A alert-type [–W *integer*$_1$ **]**
[–Q *integer*$_2$ **]**

DESCRIPTION

lpadmin configures the LP print service to describe printers and devices. It is used to add and change printers, to remove printers from the service, to set or change the system default destination, and to define alerts for print wheels.

Adding or Changing a Printer

The first form of the *lpadmin* command (**lpadmin –p** *printer options*) is used to configure a new printer or to change the configuration of an existing printer. The following options are used and may appear in any order. For ease of discussion, the printer will be referred to as *P* below.

–F *fault-recovery*
Restores the LP print service after a printer fault according to the value of *fault-recovery*:

continue Continues printing on the top of the page where printing stopped. This requires a filter to wait for the fault to clear before automatically continuing.

beginning
Starts printing the request again from the beginning.

wait Disables printing on the printer and waits for the administrator or a user to enable printing again.

During the wait, the administrator or the user who submitted the stopped print request can issue a change request that specifies where printing should resume. If no change request is made before printing is enabled, printing will resume at the top of the page where stopped if the filter allows; otherwise, the request will be printed from the beginning.

This option specifies the recovery to be used for any print request that is stopped because of a printer fault.

–c *class*
Inserts printer *P* into the specified *class*. *class* will be created if it does not already exist.

–D *comment*
Saves *comment* for display whenever a user asks for a full description of the printer *P* [see *lpstat*(1)]. The LP print service does not interpret this comment.

–e *printer*
Copies an existing *printer's* interface program to be the new interface program for printer *P*.

−f allow:*form-list*
−f deny:*form-list*
Allows (**−f allow**) or denies (**−f deny**) the forms in *form-list* to be printed on printer *P*.

For each printer, the LP print service keeps two lists of forms: an "allow-list" of forms that can be used with the printer and a "deny-list" of forms that shouldn't be used with the printer. With the **−f allow** option, the forms listed are added to the allow-list and removed from the deny-list. With the **−f deny** option, the forms listed are removed from the allow-list and added to the deny-list.

If the allow-list is not empty, the forms in the list can be used with the printer and all others cannot regardless of the content of the deny-list. If the allow-list is empty but the deny-list is not, the forms in the deny-list cannot be used with the printer. All forms can be excluded from a printer by having an empty allow-list and putting the word **any** in the deny-list. All forms can be used on a printer by having an empty deny-list and specifying **any** for the allow-list, provided the printer can handle all the characteristics of the forms.

The LP print service uses this information as a set of guidelines for determining where a form can be mounted. Administrators, however, are not restricted from mounting a form on any printer. If mounting a form on a particular printer is in disagreement with the information in the allow-list or deny-list, the administrator is warned, but the mount is accepted. Nonetheless, if a user attempts to issue a print or change request for a form and printer combination that is in disagreement with the information, the request is accepted only if the form is currently mounted on the printer. If the form is later unmounted before the request can print, the request is canceled, and the user is notified by mail.

If an administrator tries to name a form as acceptable for use on a printer that doesn't have the capabilities needed by the form, the command is rejected.

Note the other use of **−f** below.

−h Indicates that the device associated with *P* is hardwired. This option is assumed when adding a new printer unless the **−l** option is supplied.

−i *interface*
Establishes a new interface program for *P*. *interface* is the path name of the new program.

−I *content-type-list*
Assigns *P* to handle print requests with content of a type listed in *content-type-list*.

The type **simple** is recognized as the default content-type of files in the UNIX system. Such a data stream contains only printable ASCII characters and the following control characters:

Control Character	Octal Value	Meaning
backspace	10_8	move back to previous column, except at beginning of line
tab	11_8	move to next tab stop
linefeed (newline)	12_8	move to beginning of next line
form feed	14_8	move to beginning of next page
carriage return	15_8	move to beginning of current line

To force the print service to not consider **simple** as a valid type for the printer, give an explicit value (e.g., the printer type) in the *content-type-list*. If you do want **simple** included along with other types, you must include **simple** in the *content-type-list*.

Each printer automatically has its printer type included in the list of content types it will accept.

Except for **simple**, each *content-type* name is freely determined by the administrator. If names given as content types are also printer types, the names are accepted without comment because the **LP** print service recognizes all printer types as potential content types as well.

—l Indicates that the device associated with *P* is a login terminal. The **LP** scheduler, *lpsched*, disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.

—M —f *form-name* **[—a [—o filebreak]]**

Mounts the form *form-name* on *P*. Print requests to be printed with the pre-printed form *form-name* will be printed on *P*. If more than one printer has the form mounted and the user has specified any (with the **—d** option of the **lp** command) as the printer destination, then each print request will be printed on the one that meets the other needs of the request.

The page length and width and character and line pitches needed by the form are compared with those allowed for the printer by checking the capabilities in the *terminfo*(4) database for the type of printer. If the form requires attributes that are not available with the printer, the administrator is warned, but the mount is accepted. If the form lists a print wheel as mandatory but the print wheel mounted on the printer is different, the administrator is also warned but the mount is accepted.

If the **—a** option is given, an alignment pattern is printed, preceded by the same initialization of the physical printer that precedes a normal print request with one exception: no banner page is printed. Printing is assumed to start at the top of the first page of the form. After the pattern is printed, the administrator can adjust the mounted form in the printer, press return for another alignment pattern (no initialization this time), and continue printing as many alignment patterns as desired. The administrator can quit the printing alignment patterns by typing "q".

If the **—o filebreak** option is given, a formfeed is inserted between each copy of the alignment pattern. By default, the alignment pattern is assumed to correctly fill a form, so no formfeed is added.

A form is unmounted by mounting a new form in its place using the **−f** option. The **−f none** option can be used to specify no form. By default, a new printer has no form mounted.

Note the other use of **−f** above.

−M −S *print-wheel*
Mounts the print wheel *print-wheel* on *P*. Print requests to be printed with *print-wheel* will be printed on *P*. If more than one printer has the *print-wheel* mounted and the user has specified any (with the **−d** option of the **lp** command) as the printer destination, then each print request will be printed on the one that meets the other needs of the request.

If the *print-wheel* is not listed as acceptable for the printer, the administrator is warned, but the mount is accepted. If the printer does not take print wheels, the command is rejected.

A print wheel is unmounted by mounting a new print wheel in its place or by using the **−S none** option.

By default, a new printer has no special print wheel mounted. Until this is changed, a print request that asks for a specific print wheel will not be printed on *P*.

Note the other uses of the **−S** option described below.

−m *model*
Selects a model interface program provided with the LP print service for printer *P*.

−o *printing-option*
Each **−o** option in the list below is the default given to an interface program if the option is not taken from a preprinted form description or is not explicitly given by the user submitting a request [see *lp*(1)]. The only **−o** options that can have defaults defined are listed below:

length=*scaled-decimal-number*
width=*scaled-decimal-number*
cpi=*scaled-decimal-number*
lpi=*scaled-decimal-number*
stty=*stty-option-list*

The term *scaled-decimal-number* refers to a non-negative number used to indicate a unit of size. (The type of unit is shown by a trailing letter attached to the number.) Three types of scaled decimal numbers are discussed for the LP print service: numbers that show sizes in centimeters (marked with a trailing **c**), numbers that show sizes in inches (marked with a trailing **i**), and numbers that show sizes in units appropriate to use (without a trailing letter), i.e., lines, columns, lines per inch, or characters per inch.

The first four default option values should agree with the capabilities of the type of physical printer as defined in the *terminfo*(4) database for the printer type. If they do not, the command is rejected.

The *stty-option-list* is not checked for allowed values but is passed directly to the *stty*(1) program by the standard interface program. Any error messages produced by *stty*(1) when a request is processed (by the standard interface program) are mailed to the user submitting the request.

For each printing option not specified, the defaults for the following attributes are defined in the Terminfo entry for the specified printer type:

length
width
cpi
lpi

The default for **stty** is

stty=9600 cs8 −cstopb −parenb −paroff ixon
−ixany opost −olcuc −onlcr −ocrnl −onocr
−onlret −ofill nl0 cr0 tab0 bs0 vt0 ff0

You can set any of the **−o** options to the default values (which vary for different types of printers) by typing them without assigned values as follows:

length=
width=
cpi=
lpi=
stty=

−o nobanner
Allows users to submit a print request that asks that no banner page be printed.

−o banner
Forces a banner page to be printed with every print request, even when a user asks for no banner page. This is the default; you must specify **−o nobanner** if you want to allow users to specify **−o nobanner** with the **lp** command.

−r *class*
Removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then the *class* will be removed.

−S *list*
Allows the aliases for character sets or print wheels named in *list* to be used with *P*.

If the printer is a type that takes print wheels, then *list* is a list of print wheel names separated by commas or spaces. These will be the only print wheels considered mountable on the printer. (You can always force a different print wheel to be mounted, however.) Until the option is used to specify a list, no print wheels will be considered mountable on the printer, and print requests that ask for a particular print wheel with this printer will be rejected.

If the printer is a type that has selectable character sets, then *list* is a list of character set name *mappings* or aliases separated by commas or spaces. Each *mapping* is of the form

known-name = *synonym*

known-name is a character set number preceded by **cs**, such as **cs3** for character set three, or a character set name from the Terminfo database **csnm** entry. [See *terminfo*(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*] If this option is not used to specify a list, only the names already known from the Terminfo database or numbers with a prefix of cs will be acceptable for the printer.

If *list* is the word **none**, the previous print wheel list or character set aliases will be removed.

Note the other uses of the **−S** option.

−T *printer-type*
: Assigns the given *printer-type*, a representation of a physical printer of type *printer-type*. *Printer-type* is used to extract data from *terminfo*(4); this data is used to initialize the printer before printing each user's request. Some filters may also use *printer-type* to convert content for the printer. If this option is not used, the default *printer-type* will be **unknown**; no useful information will be extracted from *terminfo*(4), so each user request will be printed without first initializing the printer. Also, this option must be used if the following are to work: **−o cpi=**, **−o lpi=**, **−o width=**, and **−o length=** options of the **lpadmin** and **lp** commands, and the **−S** and **−f** options of the **lpadmin** command.

−u allow:*user-list*
−u deny:*user-list*
: Allows (**−u allow**) or denies (**−u deny**) the users in *user-list* access to *P*.

For normal access to each printer, the LP print service keeps two lists of users: an *allow-list* of people allowed to use the printer and a *deny-list* of people denied access to the printer. With the **−u allow** option, the users listed are added to the allow-list and removed from the deny-list. With the **−u deny** option, the users listed are removed from the allow-list and added to the deny-list.

If the allow-list is not empty, the users in the list are allowed access to the printer and all others are denied access, regardless of the content of the deny-list. If the allow-list is empty but the deny-list is not, the users in the deny-list are denied access and all others are allowed. If both lists are empty, all users are allowed access. Access can be denied to all users except the LP print service administrator by putting **any** in the allow-list. To allow everyone access to *P* and effectively empty both lists, put **any** in the allow-list.

−U *dial-info*
: Assigns the dialing information *dial-info* to the printer. *dial-info* is used with the *dial*(3) routine to call the printer. Any

network connection supported by the Basic Networking Utilities will work. *dial-info* can be either a phone number for a modem connection or a system name for other kinds of connections. Or if **−U direct** is given, no dialing will take place because the name **direct** is reserved for a printer that is directly connected. If a system name is given, it is used to search for connection details from the file **/usr/lib/uucp/Systems** or related files. The Basic Networking Utilities are required to support this option. By default, **−U direct** is assumed.

−v *device*

Associates a new *device* with printer *P*. *device* is the path name of a file that is writable by *lp*. Note that the same *device* can be associated with more than one printer.

−A *alert-type* [**−W** *integer*]

The **−A** option is used to send the alert *alert-type* to the administrator when a printer fault is detected and periodically thereafter until the printer fault is cleared by the administrator. The *alert-types* are

mail Sends the alert message via mail [see *mail*(1)] to the administrator who issues this command.

write Writes the message to the terminal on which the administrator is logged in. If the administrator is logged in on several terminals, one is chosen arbitrarily.

quiet Does not send messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the fault has been cleared and printing resumes, messages will again be sent when another fault occurs with the printer.

none Does not send messages until this command is given again with a different *alert-type*; removes any existing alert definition. No alert will be sent when the printer faults until a different alert-type is used (except quiet).

shell-command

shell-command is run each time the alert needs to be sent. *shell-command* should expect the message as standard input. If there are blanks embedded in the command, enclose the command in quotes. Note that the **mail** and **write** values for this option are equivalent to the values **mail** *user-name* and **write** *user-name*, respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command unless he or she has used the **su** command to change to another user ID. If the **su** command has been used to change the user ID, then the *user-name* for the new ID is used.

list The type of the alert for the printer fault is displayed on the standard output. No change is made to the alert.

The message sent appears as follows:

The print wheel *print-wheel* needs to be mounted
on the printer(s):
printer-list
number-of-requests print requests await this print-wheel.

The printer *printer-name* has stopped printing for the reason given below. Fix the problem and bring the printer back on line. Printing has stopped but will be restarted in a few minutes; issue an enable command if you want to restart sooner.

Unless someone issues a change request

lp −i *request-id* **−P** ...

to change the page-list to print, the current request will be repeated from the beginning.

The reason(s) it stopped (multiple reasons indicate reprinted attempts):

reason

The LP print service can detect printer faults only through an adequate fast filter and only when the standard interface program or a suitable customized interface program is used. Furthermore, the level of recovery after a fault depends on the capabilities of the filter.

If the *printer-name* is **all**, the alerting defined in this command applies to all existing printers.

If the **−W** option is not given or *integer*$_1$ is zero (which represents **once** and is also the default), only one message will be sent per fault. If this command is not used to arrange fault alerting for a printer, the default procedure is to mail one message to the administrator of the printer per fault.

Restrictions

When creating a new printer, either the **−v** or the **−U** option must be supplied. In addition, only one of the following may be supplied: **−e**, **−i**, or **−m**; if none of these three options are supplied, the model standard is used. The **−h** and **−l** keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters **A-Z**, **a-z**, **0-9** and **_** (underscore).

Removing a Printer Destination

The **−x** *dest* option removes the destination *dest* from the LP print service. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. If *dest* is **all**, all printers and classes are removed. No other options are allowed with **−x**.

Changing the System Default Destination

The **−d** [*dest*] option makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. No other options are allowed with **−d**.

Setting an Alert for a Print Wheel

−S *print-wheel* **−A** *alert-type* [**−W** *integer sub 1*] [**−Q** $integer_2$]

The **−S** *print-wheel* option is used with the **−A** *alert-type* option to send the alert *alert-type* to the administrator as soon as the *print-wheel* needs to be mounted and periodically thereafter. The *alert-types* are

mail Sends the alert message via mail [see *mail*(1)] to the administrator who issues this command.

write Writes the message to the terminal on which the administrator is logged in. If the administrator is logged in on several terminals, one is chosen arbitrarily.

quiet Does not send messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the *print-wheel* has been mounted and subsequently unmounted, messages will again be sent when the number of print requests again exceeds the threshold.

none Does not send messages until this command is given again with a different *alert-type* (other than **quiet**).

shell-command
The *shell-command* is run each time the alert needs to be sent. The shell command should expect the message as standard input. If there are blanks embedded in the command, enclose the command in quotes. Note that the **mail** and **write** values for this option are equivalent to the values **mail** *user-name* and **write** *user-name*, respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command unless he or she has used the **su** command to change to another user ID. If the **su** command has been used to change the user ID, then the *user-name* for the new ID is used.

list The type of the alert for the print wheel is displayed on the standard output. No change is made to the alert.

The printers listed are those that the administrator had earlier specified were candidates for this print wheel. The number ($integer_3$) listed next to each printer is the number of requests eligible for the printer. The number ($integer_4$) shown after the printer list is the total number of requests awaiting the print wheel. It will be less than the sum of the other numbers if some requests can be handled by more than one printer.

If the *print-wheel* is **all**, the alerting defined in this command applies to all print wheels already defined to have an alert.

If the **−W** option is not given or $integer_1$ is zero (which is interpreted as **once** and is also the default), only one message will be sent per need to mount a print wheel. If this command is not used to arrange alerting for a print wheel, no alerts will be sent for the print wheel.

If the **−Q** option is also given, the alert will be made when $integer_2$ print requests that need the print wheel are waiting. If the **−Q** option is not given or $integer_2$ is 1 or the word **any**, a message is sent as soon as anyone submits a print request for the print wheel when it is not mounted.

The **−S** option has a different meaning when used with the **−p** option.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpstat(1), stty(1), lpsched(1M).

dial(3), terminfo(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

lpfilter − administer filters used with the LP print service

SYNOPSIS

/usr/lib/lpfilter −f *filter-name* **−F** *path-name*
/usr/lib/lpfilter −f *filter-name* **−**
/usr/lib/lpfilter −f *filter-name* **−i**
/usr/lib/lpfilter −f *filter-name* **−x**
/usr/lib/lpfilter −f *filter-name* **−l**

DESCRIPTION

The *lpfilter* command is used to add, change, delete, and list filters used with the LP print service. These filters are used to convert the content type of a file to a content type acceptable to a given printer. One of the following options must be used with the *lpfilter* command: **−F** *path-name* (or **−** for standard input) to add or change a filter, **−i** to reset an original LP print service filter to its factory setting, **−x** to delete a filter, or **−l** to list a filter description.

The argument **all** can be used instead of a *filter-name* with any of these options. When **all** is specified with the **−F** or **−** option, the requested change is made to all filters. Using **all** with the **−i** option has the effect of restoring to their original settings all filters for which predefined settings were initially available. Using the **all** argument with the **−l** option produces a list of all filters, and using it with the **−x** option results in all filters being deleted.

Adding or Changing a Filter

The filter named in the **−f** option and described in the input is added to the filter table. If the filter already exists, its description is changed to reflect the new information in the input. Once added, a filter is available for use.

The filter description is taken from the *path-name* if the **−F** option is given or from the standard input if the **−** option is given. One of the two must be given to define or change a filter. If the filter named is one originally delivered with the LP print service, the **−i** option will restore the original filter description.

Filters are used to convert the content of a request into a data stream acceptable to a printer. For a given print request, the LP print service will know the following:

- the type of content in the request
- the name of the printer
- the type of the printer
- the types of content acceptable to the printer
- the modes of printing asked for by the originator of the request

It will use this information to find a filter that will convert the content into a type acceptable to the printer.

Below is a list of items that provide input to this command and descriptions of each item. All lists are separated by commas or spaces.

Input types: *content-type-list*
Output types: *content-type-list*
Printer types: *printer-type-list*
Printers: *printer-list*
Filter type: *filter-type*
Command: *shell-command*
Options: *template-list*

Input types
This gives the types of content that can be accepted by the filter.

Output types
This gives the types of content that the filter can produce from any of the input content types.

Printer types
This gives the type of printers for which the filter can be used. The LP print service will restrict the use of the filter to these types of printers.

Printers
This gives the names of the printers for which the filter can be used. The LP print service will restrict the use of the filter to just the printers named.

Filter type
This marks the filter as a "slow" filter or a "fast" filter. Slow filters are generally those that take a long time to convert their input. They are run unconnected to a printer to keep the printers from being tied up while the filter is running. Fast filters are generally those that convert their input quickly or those that must be connected to the printer when run. These will be given to the interface program to run connected to the physical printer.

Command
This specifies the program to run to invoke the filter. The program name as well as fixed options are included in the *shell-command*; additional options are constructed, based on the characteristics of each print request and on the **Options** field.

The command must accept a data stream as standard input and produce the converted data stream on its standard output. This allows filter pipelines to be constructed to convert data not handled by a single filter.

Options This is a list of templates separated by commas used by the LP print service to construct options to the filter from the characteristics of each print request listed in the table later. In general, each template is of the following form:

keyword pattern = *replacement*

The *keyword* names the characteristic that the template attempts to map into a filter-specific option; each valid *keyword* is listed in the table below. A *pattern* is either a literal pattern of one of the forms listed in the table or a single asterisk, *; if the *pattern* matches the value of the characteristic, the template fits and is used to generate a filter-specific option. A *pattern* of * matches any value. The *replacement* is a string used as a filter-specific option with an embedded asterisk, *, replaced with the value of the characteristic.

lp Option	Characteristic	*keyword*	Possible *patterns*
−T	Content type (input)	**INPUT**	*content-type*
N/A	Content type (output)	**OUTPUT**	*content-type*
N/A	Printer type	**TERM**	*printer-type*
−f, −o cpi=	Character pitch	**CPI**	*integer*
−f, −o lpi=	Line pitch	**LPI**	*integer*
−f, −o length=	Page length	**LENGTH**	*integer*
−f, −o width=	Page width	**WIDTH**	*integer*
−P	Pages to print	**PAGES**	*page-list*
−S	Character set/ print wheel	**CHARSET**	*character-set-name/ print-wheel-name*
−f	Form name	**FORM**	*form-name*
−y	Modes	**MODES**	*mode*
−n	Number of copies	**COPIES**	*integer*

For example, the template

MODES landscape = −l

would show that if a print request includes the **−y landscape** option, the filter should be given the option **−l**. As another example, the template

TERM * = −T *

would show that the filter should be given the option **−T** *printer-type* for whichever *printer-type* is associated with a print request using the filter.

When an existing filter is changed with this command, items that are not specified in the new information are left as they were. When a new filter is added with this command, unspecified items are given default values.

Note that a filter name and a command must be given. A filter with no input type value is assumed to work with any input type; this is also true for the output type, printer type, and printer values.

Deleting a Filter

The **−x** option is used to delete the filter specified in *filter-name* from the LP filter table.

Listing a Filter Description

The **−l** option is used to list the description of the filter named in *filter-name*. If the command is successful, the following message is sent to standard output:

Input types: *content-type-list*
Output types: *content-type-list*
Printer types: *printer-type-list*
Printers: *printer-list*
Filter type: *filter-type*
Command: *shell-command*
Options: *template-list*

If the command fails, an error message is sent to standard error.

SEE ALSO

lpadmin(1M), lp(1).

NAME

lpforms – administer forms used with the LP print service

SYNOPSIS

/usr/lib/lpforms –f *form-name option*

/usr/lib/lpforms –f *form-name* **–A** *alert-type* [–Q *integer*$_1$] [–W *integer*$_2$]

/usr/lib/lpforms –f *form-name* **–A list**

/usr/lib/lpforms –f *form-name* **–A quiet**

/usr/lib/lpforms –f *form-name* **–A none**

DESCRIPTION

The *lpforms* command is used to administer the use of preprinted forms, such as company letterhead paper, with the LP print service. The first variation of the *lpforms* command allows the administrator to add, change, and delete forms, to list the attributes of an existing form, and to allow and deny users access to particular forms. The second variation of *lpforms* is used to establish the method by which the administrator is alerted that a form must be mounted on a printer. The third variation is used to list the current alerting methods assigned to forms. The form is specified by the *form-name* given with the *lpforms* command. Users may request this form by *form-name* [see *lp*(1)]. The fourth variation of *lpforms* is to terminate an active alert. The fifth form is used to remove an alert.

With the first variation of the *lpforms* command, one of the following options must be used:

–F *path-name*	To add or change a form as specified by the information in *path-name*
–	To add or change a form, and supply information from standard input
–x	To delete a form This option must be used separately; it cannot be used with any other option.
–l	To list the attributes of a form This option must be used separately; it cannot be used with any other option.
–u allow:*user-list*	To allow users to request a form This option can be used with the **–F** or – option.
–u deny:*user-list*	To deny users access to a form This option can be used with the **–F** or – option.

Each option is explained below.

Adding or Changing a Form

The –F *path-name* option is used to add a new form to the LP print service, or to change the attributes of an existing form. The form description is taken from *path-name* if the **–F** option is given, or the standard input if the – option is given. One of the two options must be given to define or change a form. *path-name* is the path name of a file that contains all or any subset of the following information about the form.

> **Page length**: *scaled−decimal−number*$_1$
> **Page width**: *scaled−decimal−number*$_2$
> **Number of pages**: *integer*
> **Line pitch**: *scaled−decimal−number*$_3$
> **Character pitch**: *scaled−decimal−number*$_4$
> **Character set choice**: *character-set/print-wheel*,**[mandatory]**
> **Ribbon color**: *ribbon-color*
> **Comment:**
> *comment*
> **Alignment pattern**: [*content-type*]
> *content*

Except for the last two lines, the above lines can appear in any order. The **Comment:** and *comment* items must appear in consecutive order but can appear before the other items, and the **Alignment pattern:** and the *content* items must appear in consecutive order at the end of the file. Also, the *comment* item cannot contain a line that begins with any of the key phrases above, unless the key phrase is preceded with a ">" sign. Any leading > sign found in the *comment* will be removed when the comment is displayed. Case distinctions in the key phrases are ignored.

Upon issuing this command, the form named in *form-name* is added to the list of forms. If the form already exists, its description is changed to reflect the new information in the input. Once added, a form is available for use in a print request, except where access to the form has been restricted, as described under the **−u allow:** option. A form may also be allowed to be used on certain printers only.

A description of each form attribute is below:

Page length and **Page width**

Before printing the content of a print request needing this form, the generic interface program provided with the LP print service will initialize the physical printer to handle pages *scaled−decimal−number*$_1$ long, and *scaled−decimal−number*$_2$ wide using the printer type as a key into the *terminfo*(4) database. A *scaled-decimal-number* is an optionally scaled decimal number that gives a size in lines, columns, inches, or centimeters, as appropriate. The scale is indicated by appending the letter 'i', for inches, or the letter 'c', for entimeters. For length or width settings, an unscaled number indicates lines or columns; for line pitch or character pitch settings, an unscaled number indicates lines per inch or characters per inch (the same as a number scaled with 'i'). For example, **length=66** indicates a page length of 66 lines, **length=11i** indicates a page length of 11 inches, and **length=27.94c** indicates a page length of 27.94 centimeters.

The page length and page width will also be passed, if possible, to each filter used in a request needing this form.

Number of pages
Each time the alignment pattern is printed, the LP print service will attempt to truncate the *content* to a single form by, if possible, passing to each filter the page subset of 1−*integer*.

Line pitch and **Character pitch**
Before printing the content of a print request needing this form, the interface programs provided with the LP print service will initialize the physical printer to handle these pitches, using the printer type as a key into the *terminfo*(4) database. Also, the pitches will be passed, if possible, to each filter used in a request needing this form. *Scaled−decimal−number*$_3$ is in lines per centimeter if a 'c' is appended, and lines per inch otherwise; similarly, *scaled−decimal−number*$_4$ is in columns per centimeter if a 'c' is appended, and columns per inch otherwise. The character pitch can also be given as **elite** (12 characters per inch), **pica** (10 characters per inch), or **compressed** (as many characters per inch as possible).

Character set choice
When the LP print service alerts an administrator to mount this form, it will also mention that the print wheel *print-wheel* should be used on those printers that take print wheels. If printing with this form is to be done on a printer that has selectable or loadable character sets instead of print wheels, the interface programs provided with the LP print service will automatically select or load the correct character set. If **mandatory** is appended, a user is not allowed to select a different character set for use with the form; otherwise, the character set or print wheel named is a suggestion and a default only.

Ribbon color
When the LP print service alerts an administrator to mount this form, it will also mention that the color of the ribbon should be *ribbon-color*.

Comment
The LP print service will display the *comment* unaltered when a user asks about this form [see *lpstat*(1)].

Alignment pattern
When mounting this form an administrator can ask that the *content* be repeatedly printed, as an aid in correctly positioning the preprinted form. The optional *content-type* defines the type of printer for which *content* had been generated. If *content-type* is not given, **simple** is assumed. Note that the *content* is stored as given, and will be readable only by the user **lp**.

When an existing form is changed with this command, items missing in the new information are left as they were. When a new form is added with this command, missing items will get the following defaults:

Page Length: **66**
Page Width: **80**
Number of Pages: **1**

Line Pitch: **6**
Character Pitch: **10**
Character Set Choice: **any**
Ribbon Color: **any**
Comment: (no default)
Alignment Pattern: (no default)

Deleting a Form

The **−x** option is used to delete the form specified in *form-name* from the LP print service.

Listing Form Attributes

The **−l** option is used to list the attributes of the existing form specified by *form-name*. The attributes listed are those described under "Adding or Changing a Form", above. Because of the potentially sensitive nature of the alignment pattern, only the administrator can examine the form with this command. Other people can use the *lpstat*(1) command to examine the non-sensitive part of the form description.

Allowing and Denying Access to a Form

The LP print service keeps two lists of users for each form, an allow-list of people allowed to use the form, and a deny-list of people denied access to the form. With the **−u allow:** option, the users listed are added to the allow-list and removed from the deny-list. With the **−u deny:** option, the users listed are removed from the allow-list and added to the deny-list.

If the allow-list is not empty, the users in the list are allowed access to the form and all others are denied access, regardless of the content of the deny-list. If the allow-list is empty, but the deny-list is not, the users in the deny-list are denied access and all others are allowed. If both lists are empty, all users are allowed access. Access can be denied to all users, except the LP print service administrator, by putting **any** in the deny-list. To effectively empty both lists, allowing access for everyone, put **any** in the allow-list.

Alerting to Mount Forms

The second variation of the *lpforms* command is used to arrange for the alerting to mount forms on a printer.

When $integer_1$ print requests needing the preprinted form *form-name* become queued up because no printer satisfying all the needs of the requests has the form mounted, and for as long as this condition remains, an alert is sent to the administrator every $integer_2$ minutes until the form is mounted on a qualifying printer. If the *form-name* is **all**, the alerting defined in this command applies to all existing forms. No alerting is done for a backlog of print requests needing a form if the administrator does not use this option.

The method for sending the alert depends on the value of the **−A** option.

write The message is sent via *write*(1) to the terminal on which the administrator is logged in when the alert arises. If the administrator is logged in on several terminals, one is chosen arbitrarily.

mail The message is sent via mail to the administrator who issues this command.

The message sent appears as follows:

The form *form-name* needs to be mounted on the printer(s).
printer-list (*integer*$_3$ *requests*)
integer$_4$ print request awaits this form.
Use the *ribbon-color* ribbon.
Use the *print-wheel* print wheel, if appropriate.

The printers listed are those that the administrator had earlier specified were candidates for this form. The number (*integer*$_3$) listed next to each printer is the number of requests eligible for the printer. The number (*integer*$_4$) shown after the printer list is the total number of requests awaiting the form. It will be less than the sum of the other numbers if some requests can be handled by more than one printer. The *ribbon-color* and *print-wheel* are those given in the form description. The last line in the message is given even if none of the printers listed use print wheels, because the administrator may choose to mount the form on a printer that does use a print wheel.

Where any color ribbon or any print wheel can be used, the statements above will read:

Use any ribbon.
Use any print-wheel.

shell-command
The *shell-command* is run each time the alert needs to be sent. The shell command should expect the message as standard input. Note that the **mail** and **write** values for the **−A** command are equivalent to the values **mail** *user-name* and **write** *user-name*, respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command *unless* he or she has used the *su* command to change to another user ID. If the *su* command has been used to change the user ID, then the *user-name* for the new ID is used.

If the **−Q** option is not given or *integer*$_1$ is one or the word **any** (which is the default), a message is sent as soon as anyone submits a print request for the form when it is not mounted.

If the **−W** option is not given or *integer*$_2$ is zero or the word **once** (which is the default), only one message is sent when the queue size exceeds *integer*$_1$.

Listing the Current Alert

The third variation of *lpforms* is used to list the type of the alert for the specified form. No change is made to the alert. If *form-name* is recognized by the LP print service, one of the following lines is sent to the standard output, depending on the type of alert for the form.

When *integer* are queued: alert with *shell-command* every *integer* minutes

When *integer* are queued: write to *user-name* every *integer* minutes

When *integer* are queued: mail to *user-name* every *integer* minutes

No alert

The phrase "every *integer* minutes" is replaced with *once* if $integer_2$ (the **−W** $integer_2$) is 0.

Terminating an Active Alert

The **quiet** option is used to stop messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the form has been mounted and then unmounted, messages will again be sent when the queue size reaches $integer_1$ pending requests.

Removing an Alert Definition

No messages will be sent until the **none** option is given again with a different *alert-type*. This can be used to permanently stop further messages from being sent.

SEE ALSO

lp(1), lpadmin(1M).

terminfo(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

lpsched, lpshut, lpmove − start/stop the LP print service and move requests

SYNOPSIS

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove requests dest
/usr/lib/lpmove dest1 dest2

DESCRIPTION

lpsched starts the LP print service; this can be done only by **root** or **lp**.

lpshut shuts down the print service. All printers that are printing at the time *lpshut* is invoked will stop printing. When *lpsched* is started again, requests that were printing at the time a printer was shut down will be reprinted from the beginning.

lpmove moves requests that were queued by **lp** between LP destinations. The first form of the command moves the named *requests* to the LP destination *dest*. *Requests* are *request-ids* as returned by **lp**. The second form moves all requests for destination *dest1* to destination *dest2*; **lp** will then reject any new requests for *dest1*.

Note that when moving requests, *lpmove* never checks the acceptance status [see *accept*(1M)] of the new destination. Also, the request ID of the moved request is not changed so that users can still find their requests. The *lpmove* command will not move requests that have options (content type, form required, and so on) that cannot be handled by the new destination.

NOTE

By default, the directory **/usr/spool/lp** is used to hold all the files used by the LP print service. This can be changed by setting the **SPOOLDIR** environment variable to another directory before running *lpsched*. If you do this, you should populate the directory with the same files and directories found under **/usr/spool/lp**; the LP print service will not automatically create them. Also, the **SPOOLDIR** variable must then be set before any of the other LP print service commands are run.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1), lpstat(1), accept(1M), lpadmin(1M).

NAME

lpstat – print information about status of LP print service

SYNOPSIS

lpstat [options]

DESCRIPTION

lpstat prints information about the current status of the LP print service.

If no options are given, then *lpstat* prints the status of all requests made to *lp*(1) by the users. Any arguments that are not options are assumed to be *request-ids* (as returned by *lp*), printers, or printer classes. *lpstat* prints the status of such requests, printers, or printer classes. 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"

Specifying "all" after any keyletters that take "list" as an argument causes all information relevant to the keyletter to be printed. For example, the command

    lpstat –o all

prints the status of all output requests.

–a [*list*] Print acceptance status (with respect to *lp*) of destinations for requests [see *accept*(1M)]. *List* is a list of intermixed printer names and class names; the default is **all**.

–c [*list*] Print class names and their members. *List* is a list of class names; the default is **all**.

–d Print the system default destination for *lp*.

–f [*list*] [**–l**]
Print a verification that the forms in *form-list* are recognized by the LP print service. The **–l** option will list the form descriptions.

–o [*list*] [**–l**]
Print the status of output requests. *List* is a list of intermixed printer names, class names, and *request-ids*; the default is **all**. The **–l** option gives a more detailed status of the request.

–p [*list*] [**–D**] [**–l**]
Print the status of printers named in *list*. If the **–D** option is given, a brief description is printed for each printer in *list*. If the **–l** option is given, a full description of each printer's configuration is given, including the form mounted, the acceptable content and printer types, a printer description, the interface used, and so on.

–r Print the status of the LP request scheduler.

−s Print a status summary, including the system default destination, a list of class names and their members, a list of printers and their associated devices, a list of all forms currently mounted, and a list of all recognized character sets and print wheels.

−S [*list*] [**−l**]
Print a verification that the character sets or the print wheels specified in *list* are recognized by the LP print service. Items in *list* can be character sets or print wheels; the default for the list is **all**. If the **−l** option is given, each line is appended by a list of printers that can handle the print wheel or character set. The list also shows whether the print wheel or character set is mounted or specifies the built-in character set into which it maps.

−t Print all status information.

−u [*list*] Print status of output requests for users. *List* is a list of login names. The default is **all**.

−v [*list*] Print the names of printers and the path names of the devices associated with them. *List* is a list of printer names. The default is **all**.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1).

NAME

lpusers − set printing queue priorities

SYNOPSIS

/usr/lib/lpusers −d *priority-level*
/usr/lib/lpusers −q *priority-level* **−u** *user-list*
/usr/lib/lpusers −u *user-list*
/usr/lib/lpusers −q *priority-level*
/usr/lib/lpusers −l

DESCRIPTION

The *lpusers* command is used to set limits to the queue priority level that can be assigned to jobs submitted by users of the LP print service.

The first form of the command (with **−d**) sets the system-wide priority default to *priority-level*, where *priority-level* is a value of 0 to 39, with 0 being the highest priority. If a user does not specify a priority level with a print request [see *lp*(1)], the default priority is used. Initially, the default priority level is 20.

The second form of the command (with **−q** and **−u**) sets the default highest *priority-level* (0−39) that the users in *user-list* can request when submitting a print request. Users that have been given a limit cannot submit a print request with a higher priority level than the one assigned, nor can they change a request already submitted to have a higher priority. Any print requests with priority levels higher than allowed will be given the highest priority allowed.

The third form of the command (with **−u**) removes the users from any explicit priority level and returns them to the default priority level.

The fourth form of the command (with **−q**) sets the default highest priority level for all users not explicitly covered by the use of the second form of this command.

The last form of the command (with **−l**) lists the default priority level and the priority limits assigned to users.

SEE ALSO

lp(1).

NAME

ls, lc — list contents of directory

SYNOPSIS

ls [−RadCxmlnogrtucpFbqisf] [names]
lc [−RadxmlnogrtucpFbqisf] [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.

lc is a synonym for *ls* with the **−C** option set. There are three major listing formats. The default format is to list one entry per line, the **−C** and **−x** options enable multi-column formats, and the **−m** option enables stream output format. 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*(4) 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.

The *ls* command has the following options:

−R Recursively list subdirectories encountered.

−a List all entries, including those that begin with a dot (.), which are normally 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 Multi-column output with entries sorted down the columns (default setting for *lc*).

−x Multi-column output with entries sorted across rather than down the page.

−m Stream output format; files are listed across the page, separated by commas.

−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 stamp (latest first) instead of by name. The default is the last modification time. (See **−n** and **−c**.)

−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 inode (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-printable characters to be in the octal **\ddd** notation.

−q Force printing of non-printable characters in file names as the character question mark (?).

−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 ten characters. The first character may be one of the following:

- **d** the entry is a directory;
- **b** the entry is a block special file;
- **c** the entry is a character special file;
- **m** the entry is a XENIX shred data (memory) file;
- **p** the entry is a fifo (a.k.a. "named pipe") special file;
- **s** the entry is a XENIX semaphore;
- **−** 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.

ls −l (the long list) prints its output as follows:

```
−rwxrwxrwx  1 smith  dev    10876  May 16 9:42 part2
```

This horizontal configuration provides a good deal of information. Reading from right to left, you see that the current directory holds one file, named "part2." Next, the last time that file's contents were modified was 9:42 A.M. on May 16. The file is moderately sized, containing 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group "dev" (perhaps indicating "development"), and his or her login name is "smith." The number, in this case "1," indicates the number of links to file "part2." Finally, the row of dash and letters tell you that user, group, and others have permissions to read, write, execute "part2."

The execute (**x**) symbol here occupies the third position of the three-character sequence. A – in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

- **r** the file is readable
- **w** the file is writable
- **x** the file is executable
- **–** the indicated permission is *not* granted
- **l** mandatory locking will occur during access (the set-group-ID bit is on and the group execution bit is off)
- **s** the set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on
- **S** undefined bit-state (the set-user-ID bit is on and the user execution bit is off)
- **t** the 1000 (octal) bit, or sticky bit, is on [see *chmod*(1)], and execution is on
- **T** the 1000 bit is turned on, and execution is off (undefined bit-state)

For user and group permissions, the third position is sometimes occupied by a character other than **x** or **–**. **s** also may occupy this position, referring to the state of the set-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as root but need to assume the identity of the user stated at "login."

In the case of the sequence of group permissions, **l** may occupy the third position. **l** refers to mandatory file and record locking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For others permissions, the third position may be occupied by **t** or **T**. These refer to the state of the sticky bit and execution permissions.

EXAMPLES

An example of a file's permissions is:

–rwxr––r––

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

Another example of a file's permissions is:

–rwsr–xr–x

This describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

Another example of a file's permissions is:

–rw–rwl–––

This describes a file that is readable and writable only by the user and the group and can be locked during access.

An example of a command line:

ls −a

This command will print the names of all files in the current directory, including those that begin with a dot (.), which normally do not print.

Another example of a command line:

ls −aisn

This command will provide you with quite a bit of information including **a**ll files, including non-printing ones (**a**), the **i**-number—the memory address of the inode associated with the file—printed in the left-hand column (**i**); the **s**ize (in blocks) of the files, printed in the column to the right of the i-numbers (**s**); finally, the report is displayed in the **n**umeric version of the long list, printing the **UID** (instead of user name) and **GID** (instead of group name) numbers associated with the files.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	user IDs for **ls −l** and **ls −o**
/etc/group	group IDs for **ls −l** and **ls −g**
/usr/lib/terminfo/?/*	terminal information database

SEE ALSO

chmod(1), find(1).

BUGS

Unprintable characters in file names may confuse the columnar output options.

NAME

machid: i386 – get processor type truth value

SYNOPSIS

i386

DESCRIPTION

The following commands will return a true value (exit code of 0).

The commands that do not apply will return a false (non-zero) value. These commands are often used within makefiles [see *make*(1)] and shell procedures [see *sh*(1)] to increase portability.

SEE ALSO

sh(1), test(1), true(1).
make(1) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

mail, rmail – send mail to users or read mail

SYNOPSIS

Sending mail:

mail [**–wt**] persons

rmail [**–wt**] persons

Reading mail:

mail [**–ehpqr**] [**–f** file] [**–F** persons]

DESCRIPTION

Sending mail:

The command-line options that follow affect SENDING mail:

–w causes a letter to be sent to a remote user without waiting for the completion of the remote transfer program.

–t causes a **To:** line to be added to the letter, showing the intended recipients.

A *person* is usually a user name recognized by *login*(1). When *persons* are named, *mail* assumes a message is being sent (except in the case of the **–F** option). It reads from the standard input up to an end-of-file (control-d), or until it reads a line consisting of just a period. When either of those signals is received, *mail* adds the *letter* to the *mailfile* for each *person*. A *letter* is a *message* preceded by a *postmark*. The message is preceded by the sender's name and a *postmark*. A *postmark* consists of one or more 'From' lines followed by a blank line.

If a letter is found to be undeliverable, it is returned to the sender with diagnostics that indicate the location and nature of the failure. If *mail* is interrupted during input, the file **dead.letter** is saved to allow editing and resending. **dead.letter** is recreated every time it is needed, erasing any previous contents.

rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

If the local system has the Basic Networking Utilities installed, mail may be sent to a recipient on a remote system. Prefix *person* by the system name and exclamation point. A series of system names separated by exclamation points can be used to direct a letter through an extended network.

Reading Mail:

The command-line options that follow affect READING 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.

–h causes a window of headers to be displayed rather than the latest message. The display is followed by the ? prompt.

–p causes all messages to be printed without prompting for disposition.

–q causes *mail* to terminate after interrupts. Normally an interrupt causes only 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*.

−F*persons*
entered into an empty *mailbox*, causes all incoming mail to be forwarded to *persons*.

mail, unless otherwise influenced by command-line options, prints a user's mail messages in last-in, first-out order. For each message, the user is prompted with a ?, and a line is read from the standard input. The following commands are available to determine the disposition of the message:

<new-line>, **+**, or **n**	Go on to next message.
d or **dp**	Delete message and go on to next message.
d #	Delete message number #. Do not go on to next message.
dq	Delete message and quit *mail*.
h	Display a window of headers around current message.
h #	Display header of message number #.
h a	Display headers of ALL messages in the user's *mailfile*.
h d	Display headers of messages scheduled for deletion.
p	Print current message again.
−	Print previous message.
a	Print message that arrived during the *mail* session.
#	Print message number #.
r [*users*]	Reply to the sender, and other *user(s)*, then delete the message.
s [*files*]	Save message in the named *files* (**mbox** is default).
y	Same as save.
u [#]	Undelete message number # (default is last read).
w [*files*]	Save message, without its top-most header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons*.
q, or **ctl-d**	Put undeleted mail back in the *mailfile* and quit *mail*.
x	Put all mail back in the *mailfile* unchanged and exit *mail*.
!*command*	Escape to the shell to do *command*.

? Print a command summary.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

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*. A "Forwarded by..." message is added to the header. This is especially useful in a multi-machine environment to forward all of a person's mail to a single machine and to keep the recipient informed if the mail has been forwarded. Installation and removal of forwarding is done with the **−F** option.

To forward all of one's mail to systema!user enter:

mail −Fsystema!user

To forward to more than one user enter:

mail −F"user1,systema!user2,systema!systemb!user3"

Note that when more than one user is specified, the whole list should be enclosed in double quotes so that it may all be interpreted as the operand of the **−F** option. The list can be up to 1024 bytes; either commas or white space can be used to separate users.

The following list of characters are prohibited from appearing anywhere in the *mail* **−F** argument list or in the "Forward to" line:

; & | ^ < > ‘ ( ) <CR>

To remove forwarding enter:

mail −F ""

The pair of double quotes is mandatory to set a NULL argument for the **−F** option.

For forwarding to work properly, the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*user*	incoming mail for *user*; i.e., the *mailfile*
$HOME/mbox	saved mail
$MAIL	variable containing path name of *mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), mailx(1), write(1).

WARNINGS

The "Forward to person" feature may result in a loop if *sys1!userb* forwards to *sys2!userb* and *sys2!userb* forwards to *sys1!userb.* The symptom is a message saying "unbounded... saved mail in dead.letter."

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.

Many of the remote features of *mailx* will only work if the Basic Networking Utilities are installed on your system.

Incoming mail is stored in a standard file for each user, called the **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 can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the **–f** option of the *mailx* command. Messages in the secondary file can then be read or otherwise processed using the same COMMANDS as in the primary **mailbox**. This gives rise within these pages to the notion of a current **mailbox**.

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:

–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).
–h *number*	The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops. (See **addsopt** under ENVIRONMENT VARIABLES).
–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. (See **addsopt** under ENVIRONMENT VARIABLES).
−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. (See **addsopt** under ENVIRONMENT VARIABLES).

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. (A "subject" longer than 1024 characters will cause *mailx* to dump core.) 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 mail is found to to undeliverable, an attempt is made to return it to the sender's *mailbox*. 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 right angle bracket (>) in the header summary. Many commands take an optional list of messages (*msglist*) to operate on. The default for *msglist* is the current message. A *msglist* is a list of message identifiers 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* tries to execute commands from the optional system-wide file (**/usr/lib/mailx/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. With the exceptions noted below, regular commands are legal inside start-up files. The most common use of a start-up file is 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. An error in the start-up file causes the remaining lines in the file to be ignored. The **.mailrc** file is optional and must be constructed locally.

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** 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 **F**ollowup, **S**ave, and **C**opy 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 **fo**llowup, **S**ave, and **C**opy 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* | *r*
s
else
s
endif
Conditional execution, where *s* will execute following s, up to an **else** or **endif**, if the program is in *send* mode, and *r* causes the s 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 **P**rint and **T**ype commands override this command.

list
Prints all commands available. No explanation is given.

mail *name* ...
Mail a message to the specified users.

Mail *name*
Mail a message to the specified user and record a copy of it in a file named after that user.

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 **ex**it and **q**uit 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 file name, 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 file name, 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**, or the file specified in the **MBOX** environment variable, 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**).

z[+ | −]

Scroll the header display forward or backward one full screen. 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).

~:
~_

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.

~h
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.' Environment variables set and exported in the shell are also accessible by ~i.

~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.

~v

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.

addsopt

Enabled by default. If **/bin/mail** is not being used as the deliverer, **noaddsopt** should be specified. (See WARNINGS below)

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 **alt**ernates command and the **metoo** variable.

append
Upon termination, append messages to the end of the **mbox** 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 **d**elete and **u**ndelete 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 **pi**pe 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. 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. Takes effect with next message sent.

folder=*directory*
The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. 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, **fo**llowup, and Followup commands.

page
Used with the **pi**pe command to insert a form feed after each message sent through the pipe. Default is **nopage**.

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 full screen of headers for the **h**eaders command.

sendmail=*shell-command*
Alternate command for delivering messages. Default is */bin/rmail*(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	optional global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO

ls(1), mail(1), pg(1).

WARNINGS

The **−h**, **−r** and **−U** options can be used only if *mailx* is built with a delivery program other than */bin/mail*.

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).

NAME

man – print entries in this manual

SYNOPSIS

man [options] [section] titles

DESCRIPTION

The *man* command locates and prints the entry of this manual named title in the specified section. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) The *title* is entered in lowercase. The *section* number may not have a letter suffix. If no *section* is specified, the whole manual is searched for title and all occurrences of it are printed. The *options* and their meanings are:

–T*term* Print the entry as appropriate for terminal type *term*. For a list of recognized values of *term*, type **help term2**.

–w Print on the standard output only the path names of the entries, relative to **/usr/man**, or to the current directory for the **–d** option.

–d Search the current directory rather than **/usr/catman**; this requires the full file name (e.g., *cu.1c*, rather than just *cu*).

–c Cause man to invoke *col*(1); note that *col*(1) is invoked automatically by *man* if the INTERACTIVE Text Processing Workbench is installed.

The *man* command examines the environment variable $TERM (see *environ*(5)) and attempts to select options that adapt the output to the terminal being used. The –T*term* option overrides the value of $TERM; in particular, one should use –T*lp* when sending the output of *man* to a line printer.

The *section* may be changed before each *title*.

As an example:

```
man man
```

would display this entry, as well as any other entries named *man* that may exist in other sections of the manual, on the terminal.

FILES

/usr/catman/? man/man[1-8]* Preformatted manual entries

SEE ALSO

environ(5), term(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

WARNING

The *man* command prints manual entries that were formatted using *nroff* when the UNIX System was installed. Entries are originally formatted with terminal type **37**, and are printed using the correct terminal filters as derived from the –T*term* and $TERM settings. Typesetting or other non-standard printing of manual entries requires that the INTERACTIVE Text Processing Workbench be installed.

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [−n] [−y]

DESCRIPTION

The *mesg* command with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. The *mesg* command 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 directories

SYNOPSIS

mkdir [**−m** mode] [**−p**] dirname ...

DESCRIPTION

The *mkdir* command creates the named directories in mode 777 [possibly altered by *umask*(1)].

Standard entries in a directory (e.g., the files ., for the directory itself, and .., for its parent) are made automatically. *mkdir* cannot create these entries by name. Creation of a directory requires write permission in the parent directory.

The owner ID and group ID of the new directories are set to the process's real user ID and group ID, respectively.

Two options apply to *mkdir*:

−m This option allows users to specify the mode to be used for new directories. Choices for modes can be found in *chmod*(1).

−p With this option, *mkdir* creates *dirname* by creating all the non-existing parent directories first.

EXAMPLE

To create the subdirectory structure **ltr/jd/jan**, type:

```
mkdir -p ltr/jd/jan
```

SEE ALSO

sh(1), rm(1), umask(1).
intro(2), mkdir(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

DIAGNOSTICS

The *mkdir* command returns exit code 0 if all directories given in the command line were made successfully. Otherwise, it prints a diagnostic and returns non-zero. An error code is stored in *errno*.

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs special blocks[:inodes] [gap blocks/cyl] [**–b** blocksize]
/etc/mkfs special proto [gap blocks/cyl] [**–b** blocksize]

DESCRIPTION

mkfs constructs a file system by writing on the *special* file using the values found in the remaining arguments of the command line. The command waits 10 seconds before starting to construct the file system. During this 10-second pause the command can be aborted by entering a delete (DEL).

The **–b** *blocksize* option specifies the logical block size for the file system. The logical block size is the number of bytes read or written by the operating system in a single I/O operation. Valid values for *blocksize* are 512, 1024, and 2048. The default is 1024. A block size of 2048 may be chosen only if the 2K file system package is installed. If the **–b** option is used, it must appear last on the command line.

If the second argument to *mkfs* is a string of digits, the size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* (512-byte) disk blocks the file system will occupy. If the number of inodes is not given, the default is approximately the number of *logical* blocks divided by 4. *mkfs* builds a file system with a single empty directory on it. The boot program block (block zero) is left uninitialized.

If the second argument is the name of a file 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. A sample prototype specification follows (line numbers have been added to aid in the explanation):

```
1.    /stand/ diskboot
2.    4872 110
3.    d––777 3 1
4.    usr    d––777 3 1
5.           sh     –––755 3 1 /bin/sh
6.           ken    d––755 6 1
7.                  $
8.           b0     b––644 3 1 0 0
9.           c0     c––644 3 1 0 0
10.          $
11.   $
```

Line 1 in the example is the name of a file to be copied onto block zero as the bootstrap program.

Line 2 specifies the number of *physical* (512-byte) blocks the file system is to occupy and the number of inodes in the file system.

Lines 3-9 tell *mkfs* about files and directories to be included in this file system.

Line 3 specifies the root directory.

Lines 4-6 and 8-9 specify other directories and files.

The $ on line 7 tells *mkfs* to end the branch of the file system it is on, and continue from the next higher directory. The $ on lines 10 and 11 end the process, since no additional specifications follow.

File specifications give the mode, the user ID, the group ID, and the initial contents of the file. Valid syntax for the contents field depends on the first character of the mode.

The mode for a file is specified by a 6-character string. The first character specifies the type of the file. The character range is **−bcd** to specify regular, block special, character special and directory files, respectively. The second character of the mode 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 of the specification may be a path name from which the contents and size are copied. If the file is a block or character special file, two decimal numbers 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) file specifications for the entries in the directory. As noted above, the scan is terminated with the token **$**.

The *gap blocks/cyl* argument in both forms of the command specifies the rotational gap and the number of blocks/cylinder.

FILES

/etc/vtoc/*

SEE ALSO

chmod(1).
dir(4), fs(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

BUGS

With a prototype file, it is not possible to copy in a file larger than 64K bytes, nor is there a way to specify links. The maximum number of inodes configurable is 65500.

NAME

mknod — build special file

SYNOPSIS

/etc/mknod name **b** | **c** major minor
/etc/mknod name **p**

DESCRIPTION

The *mknod* command makes a directory entry and corresponding inode for a special file.

The first argument is the *name* of the entry. The UNIX System convention is to keep such files in the **/dev** directory.

In the first case, the second argument 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). They may be either decimal or octal. The assignment of major device numbers is specific to each system. The information is contained in the system source file **conf.c**. You must be the super-user to use this form of the command.

The second case is the form of the *mknod* that is used to create FIFO's (a.k.a named pipes).

WARNING

If *mknod* is used to create a device in a remote directory (Remote File Sharing), the major and minor device numbers are interpreted by the server.

SEE ALSO

mknod(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

mkpart — disk maintenance utility

SYNOPSIS

/etc/mkpart [**−f** filename] [**−p** partition] ... [**−P** partition] ...
[**−b**] [**−B** filename] [**−A** sector] ... [**−V**] [**−v**]
[**−i** [**−I** interleave]] [**−x** file] [**−t** [**vpa**]]
[**−n**] [**−O**] device

DESCRIPTION

This program allows the system administrator to display and modify the data structures that the disk driver uses to access disks. These structures describe the number, size, and type of the partitions, as well as the physical characteristics of the disk drive itself.

The user maintains a file of stanzas, each of which contains comments and parameters. The stanzas are of two varieties: those that describe disk partitions and disk devices. Stanzas may refer to other stanzas of the same type so that common device or partition types may be customized. By default, the stanza file is named **/etc/partitions**. The required parameter *device* specifies the device stanza for the disk to be used.

The following options can be used with *mkpart*:

−f *filename*
: Specifies the partition and device specification stanza file. If not present, **/etc/partitions** is assumed.

−p *partition*
: Removes a partition from the *vtoc* on the specified device. The *partition* is a stanza that indicates the partition to be removed by its partition number parameter; no comparisons are made by attribute. Note that alternate partitions cannot be removed.

−P *partition*
: Adds a partition to the *vtoc* on the specified device. A *partition* is a stanza that contains and/or refers to other stanzas that contain all of the necessary parameters for a *vtoc* partition.

−b
: Causes only the *boot* program to be updated, unless other options are specified.

−B *filename*
: Specifies a different *boot* program than the one given by the device stanza.

−F *interleave*
: Causes the entire device to be hardware formatted. This process rewrites all the sector headers on each track of the disk, enabling subsequent access using normal *read*s and *write*s. *interleave* is the distance in physical sectors between each successive logical sector. Normal values are *1* for track-cache controllers and *2-4* for standard controllers.

−A *sector*
: Marks the specified sector as bad and assigns it an alternate if possible. *sector* is a zero-based absolute sector number from

the beginning of the drive. To compute a sector number given cylinder, head, and (zero-based) sector in track, the formula is:

cylinder * (sectors-per-track * heads-per-cylinder) + head * (sectors-per-track) + sector

−V Causes a complete surface-analysis pass to be run. This first writes a data pattern (currently 0xe5 in every byte) to each sector of the disk, then reads each sector. Any errors are noted and the bad sectors found are added to the alternates table if possible.

−v Causes a non-destructive surface-analysis pass to be run. This pass reads every sector of the disk, noting bad sectors as above.

−i Initializes the Volume Table of Contents (VTOC) on the drive to default values, clearing any existing partition and bad-sector information which may have existed. This is the only way to remove an alternate partition and can be used to re-initialize a drive which may have obsolete or incorrect VTOC data on it.

−I *interleave*
Provides an interleave value to use to initialize the VTOC without actually formatting the drive.

−x *file* Writes a complete *device* and *partition* stanza list for the specified *device* to *file*.

−t [*vpa*]
Creates a listing of the current *vtoc*. The sub-parameters specify pieces to be printed:

v *vtoc* and related structures
p partitions
a alternate sectors

−n Turns on a special mode for message compatibility with the installation. Other than the output format, no functionality is changed. This option is not recommended for normal use.

−O Forces the creation of an "old style" PDINFO. This is a debugging option only; its use can destroy the disk structure integrity.

The partitions file is composed of blank-line-separated stanzas. (Blank lines have only tabs and spaces between new-line characters.) Commentary consists of all text between a # and a new-line character. Stanzas begin with an identifier followed by a colon (:), and are followed by a comma-separated list of parameters. Each parameter has a keyword followed by an equal sign (=) followed by a value. The value can be a number, another stanza's name, a double quoted string, or a parentheses-surrounded, comma-separated list of numbers or ranges of numbers, as appropriate for the keyword. Numbers may be written as decimal, octal, or hexadecimal constants in the form familiar to C programmers.

Device specification stanzas may contain the following parameters:

usedevice = *name*
Causes the named stanza's parameters to be included in the device definition.

boot = *string*
Indicates that the string is the filename of a bootstrap program to install on the disk.

device = *string*
Gives the filename of the character special device for the disk.

heads = *number*
Specifies the number of tracks per cylinder on the device.

cyls = *number*
Is the number of cylinders on the disk.

sectors = *number*
Is the number of sectors per track.

bpsec = *number*
Is the number of bytes per sector.

physheads, physcyls, physsectors, physbpsec = *number*
Are the physical values associated with the disk, if they are different from those values used in normal operation. This situation occurs with disk controllers that support sector sparing and/or virtual geometries. These values are needed to properly format disks connected to such controllers.

dserial = *string*
Is an arbitrary string which is recorded in the volume label (Multibus systems only).

vtocsec = *number*
Gives the sector number to use for the VTOC. Note that for AT386 systems, this number *must* be **17**.

altsec = *number*
Is the sector to use for the alternate block table.

badsec = *number-list*
Lists the known bad sectors. These are appended to any specified in the command line or found during surface analysis.

Partition stanzas may have the following parameters:

usepart = *name*
Refers to another partition stanza.

partition = *number*
Gives this partition's entry number in the *vtoc*.

tag = *tagname*
A partition tag specifies the purpose of the partition. The *tagnames* are reserved words which are presently used for identification purposes *only*:

BACKUP means the entire disk.

ROOT is a **root** file system partition.

BOOT is a bootstrap partition.

SWAP is a partition that does not contain a file system.

USR is a partition that does contain a file system.

ALTSCTR is an alternate sector/track partition that contains four sections. It permits a variable number of replacement sectors with no inherent maximum.

1. The partition table section, located at the beginning of the first sector of the partition, contains information about the general layout of this partition.
2. The partition map section is a bitmap of all the sectors allocated in the alternate sector/track partition. It identifies sectors that are either good or bad/allocated.
3. The entry section contains a list of bad sector entries. Each entry includes the starting bad sector number, its count and the starting alternate sector number. Contiguous bad sectors that are added at the same time are grouped into one single entry as a cluster of bad sectors.
4. The reserved section is a pool of alternate sectors used for bad sector remap.

ALTS Contains alternate sectors to which the driver remaps bad sectors. Currently, a maximum of 253 alternate sectors are supported.

OTHER
Is a partition that the UNIX System does not know how to handle, such as MS-DOS space.

perm = *permname*
Specifies a permission type for the partition. Permissions are not mutually exclusive:

RO indicates that the partition cannot be written upon. Normally, write access is granted (standard UNIX System file permissions notwithstanding).

NOMOUNT disallows the driver from mounting the file system that may be contained in the partition.

VALID indicates that the partition contains valid data. Any partition added with the **−A** flag will be marked VALID.

start = *number*
Is the starting sector number for the partition. Note that for AT386 systems, the **root** file system should start at the *second* track of the cylinder, which is the beginning of the active UNIX System *fdisk* partition. This allows space for the writing of the *boot* code.

size = *number*
Is the size, in sectors, of the partition.

When *mkpart* is run, it first attempts to read the volume label (for Multibus systems) or the *fdisk* table (for AT386 systems), the *vtoc* block, and the alternate sector table. If any of the structures is invalid or cannot be read, or if the **−i** flag is specified, the internal tables are initialized to default values for the device specified (taken from the device stanza in the partition file). If the **−F** flag is specified, the device is formatted. If either the **−V** or **−v** flag is specified, the appropriate surface analysis is performed.

After these steps, partitions are deleted or added as required. Next, any bad sectors specified in the partition file, found during surface analysis, or specified in the command line with **−A** flags are merged into the alternate sectors table. Note that an alternate partition must exist for any bad-sector marking to occur, as bad sectors are assigned good alternates at this point.

Finally, the *boot* program is written to track 0 of cylinder 0 (Multibus systems) or the cylinder where the active UNIX System *fdisk* partition starts (AT386 systems). If **−b** was not the only parameter specified, the updated VTOC is written, and and the disk driver is instructed to re-read the tables when the drive is opened the next time. Bad sectors that are found during the disk preparation procedure are stored in the alternate tables during the creation of the alternate sector/track partition. If bad sectors are added using the **−A** option during the course of normal system operation, the tables in the alternate sector/track partition are updated. In addition, the disk driver will be instructed to update its internal alternate sector/track tables. This allows grown defect sectors to be remapped right away without closing the drive or shutting down the system.

When only **−t** is specified, only a listing is created and no updating occurs.

FILES

/etc/partitions
/etc/boot
/dev/rdsk/*p0

BUGS

Currently, very little consistency checking is done. No checks are made to ensure that the *fdisk* partition table is consistent with the UNIX System partitions placed in the VTOC.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

more – view a file one full screen at a time

SYNOPSIS

more [–cdflsurw] [–*n*] [+*linenumber*] [+/*pattern*] [*name* ...]

DESCRIPTION

The *more* filter allows examination of continuous text one full screen at a time. It normally pauses after each full screen, printing "--More--" at the bottom of the screen. If the user then presses the RETURN key, one more line is displayed. If the user presses the SPACEBAR, another full screen is displayed. Other possibilities are described below.

The command line options include the following:

–*n* An integer which is the size (in lines) of the window which *more* will use instead of the default.

–c *More* draws each page by beginning at the top of the screen and erasing each line just before it displays a new line. This avoids scrolling the screen, making it easier to read while *more* is writing. This option is ignored if the terminal does not have the ability to clear to the end of a line.

–d *More* prompts with the message "Hit space to continue, Del to abort" at the end of each full screen. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be inexperienced.

–f This option causes *more* to count logical lines, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters that would ordinarily occupy screen positions, but that do not print when they are sent to the terminal as part of an escape sequence. Thus, *more* may think that lines are longer than they actually are and fold lines erroneously.

–l Does not treat CTRL-L (FORMFEED) specially. If this option is not given, *more* pauses after any line that contains a CTRL-L, as if the end of a full screen had been reached. Also, if a file begins with a FORMFEED, the screen is cleared before the file is printed.

–s Squeezes multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

–u Normally, *more* handles underlining, such as that produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* outputs appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The **−u** option suppresses this processing.

−r Normally, *more* ignores control characters that it does not interpret in some way. The **−r** option causes these to be displayed as ^C, where "C" stands for any such control character.

−w Normally, *more* exits when it comes to the end of its input. With **−w**, however, *more* prompts and waits for any key to be struck before exiting.

+*linenumber* Starts up at *linenumber*.

+/*pattern* Starts up two lines before the line containing the regular expression *pattern*.

More looks in the **/etc/termcap** file for the terminal characteristics and the default window size. For example, on a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable, MORE, to preset any flags desired. For example, if you prefer to view files using the **−c** mode of operation, the shell command, "MORE=−c", in the **.profile** file causes all invocations of *more* to use this mode.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the "--More--" prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1):

i <*space*> Displays *i* more lines, (or another full screen if no argument is given).

CTRL-D Displays 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d Same as CTRL-D.

*i***z** Same as typing a space except that *i*, if present, becomes the new window size.

*i***s** Skips *i* lines and prints a full screen of lines.

*i***f** Skips *i* full screens and prints a full screen of lines.

q or **Q** Exits from *more*.

= Displays the current line number.

v Starts up the screen editor *vi* at the current line. (Note that *vi* may not be available with your system.)

h or **?** Help command; gives a description of all the *more* commands.

i/*expr*	Searches for the *i*th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a full screen is displayed, starting two lines before the place where the expression was found. You can use the ERASE and KILL characters to edit the regular expression. Erasing back past the first column cancels the search command.
*i***n**	Searches for the *i*th occurrence of the last regular expression entered.
'	(Single quotation mark) Goes to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.
!*command*	Invokes a shell with *command*. The % and ! characters in *command* are replaced with the current filename and the previous shell command respectively. If there is no current filename, % is not expanded. The sequences, "\%" and "\!", are replaced by "%" and "!", respectively.
*i***:n**	Skips to the *i*th next file given in the command line (skips to last file if n doesn't make sense).
*i***:p**	Skips to the *i*th previous file given in the command line. If this command is given in the middle of printing out a file, *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell rings and nothing else happens.
:f	Displays the current filename and line number.
:q or **:Q**	Exits from *more* (same as *q* or *Q*).
.	Repeats the previous command.

The commands take effect immediately, i.e., you do not need to press the RETURN key. Up to the time when the command character itself is given, you can press the KILL character to cancel the numerical argument being formed. In addition, you can press the ERASE character to redisplay the "--More--(*xx*%)" message.

The terminal is set to *noecho* mode by this program so that the output can be continuous. This means that what you type does not show on your terminal, except for the slash (/) and exclamation (!) commands.

If the standard output is not a teletype, *more* acts just like *cat*, except that a header is printed before each file (if there is more than one file).

A sample usage of *more* in previewing *nroff* output would be:

```
nroff −ms +2 doc.n | more -s
```

FILES

/usr/lib/terminfo/?/*	terminal information database
/usr/lib/more.help	help file

SEE ALSO

csh(1), sh(1).

environ(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NOTES

The *vi* and *help* options may not be available.

Before displaying a file, *more* attempts to detect whether the file is an unprintable binary file such as a directory or an executable binary image. If *more* concludes that a file is unprintable, *more* refuses to print it. However, *more* cannot detect all possible kinds of unprintable files.

This utility was developed at the University of California at Berkeley and is used with permission.

NAME

mount, umount − mount and unmount file systems and remote resources

SYNOPSIS

/etc/mount [**−r**] [**−f** fstyp] special directory
/etc/mount [**−r**] [**−c**] **−d** resource directory
/etc/mount
/etc/umount special
/etc/umount **−d** resource

DESCRIPTION

File systems other than **root** (/) are considered *removable* in the sense that they can be either available to users or unavailable. *mount* announces to the system that *special*, a block special device or *resource*, a remote resource, is available to users from the mount point *directory*. *directory* must exist already; it becomes the name of the root of the newly mounted *special* or *resource*. A unique resource may be mounted only once (no multiple mounts).

mount, when entered with arguments, adds an entry to the table of mounted devices, **/etc/mnttab**. *umount* removes the entry. If invoked with no arguments, *mount* prints the entire mount table. If invoked with any of the following partial argument lists, *mount* will search **/etc/fstab** to fill in the missing arguments: *special*, **−d** *resource*, *directory*, or **−d** *directory*.

The following options are available:

−r indicates that *special* or *resource* is to be mounted read-only. If *special* or *resource* is write-protected or read-only advertised, this flag must be used.

−d indicates that *resource* is a remote resource that is to be mounted on *directory* or unmounted. To mount a remote resource, Remote File Sharing must be up and running and the resource must be advertised by a remote computer [see *rfstart*(1M) and *adv*(1M)]. If **−d** is not used, *special* must be a local block special device.

−c indicates that remote reads and writes should not be cached in the local buffer pool. **−c** is used in conjunction with **−d**.

−f *fstyp* indicates that *fstyp* is the file system type to be mounted. If this argument is omitted, it defaults first to the **root** *fstyp*, then, if it cannot be mounted as that type but it is a local file system, it will attempt to determine the *fstyp* by executing **/etc/fstyp**.

special indicates the block special device that is to be mounted on *directory*.

resource indicates the remote resource name that is to be mounted on a *directory*.

directory indicates the directory mount point for *special* or *resource*. (The directory must already exist.)

umount announces to the system that the previously mounted *special* or *resource* is to be made unavailable. If invoked with *directory* or **−d** *directory*, *umount* will search **/etc/fstab** to fill in the missing argument(s).

mount can be used by any user to list mounted file systems and resources. Only a super-user can mount and unmount file systems.

Example

To mount a DOS file system on a system where DOS resides on the first *fdisk* partition, type:

mount [−f DOS] /dev/dsk/0p1 /mnt

FILES

/etc/mnttab	mount table
/etc/fstab	file system table

SEE ALSO

adv(1M), fuser(1M), fstyp(1M), nsquery(1M), rfstart(1M), rmntstat(1M), setmnt(1M), unadv(1M).

mount(2), umount(2), fstab(4), mnttab(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

INTERACTIVE Network Connection Facilities Guide for guidelines on mounting remote resources.

DIAGNOSTICS

If the *mount*(2) system call fails, *mount* prints an appropriate diagnostic. *mount* issues a warning if the file system to be mounted is currently labeled under another name. A remote resource mount will fail if the resource is not available or if Remote File Sharing is not running or if it is advertised read-only and not mounted with **−r**.

umount fails if *special* or *resource* is not mounted or if it is busy. *special* or *resource* is busy if it contains an open file or some user's working directory. In such a case, you can use *fuser*(1M) to list and kill processes that are using *special* or *resource.*

WARNINGS

Physically removing a mounted file system diskette from the diskette drive before issuing the *umount* command damages the file system.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

mountall, umountall – mount, unmount multiple file systems

SYNOPSIS

/etc/**mountall** [–] [file-system-table] ...
/etc/**umountall** [**–k**]

DESCRIPTION

These commands may be executed only by the super-user.

The *mountall* command is used to mount file systems according to a *file-system-table*. (**/etc/fstab** is the default file system table.) The special file name "–" reads from the standard input.

Before each file system is mounted, it is checked using *fsstat*(1M) to see if it appears mountable. If the file system does not appear mountable, it is checked, using *fsck*(1M), before the mount is attempted.

The *umountall* command causes all mounted file systems except **root** to be unmounted. The **–k** option sends a SIGKILL signal, via *fuser*(1M), to processes that have files open.

FILES

File-system-table format:

column 1	block special file name of file system
column 2	mount-point directory
column 3	"–r" if to be mounted read-only; "–d" if remote
column 4	(optional) file system type string
column 5+	ignored

White space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

A typical file-system-table might read:

/dev/dsk/0s1 /usr –r S51K

SEE ALSO

fsck(1M), fsstat(1M), fuser(1M), mount(1M).

signal(2), fstab(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

DIAGNOSTICS

No messages are printed if the file systems are mountable and clean.

Error and warning messages come from *fsck*(1M), *fsstat*(1M), and *mount*(1M).

NOTES

The information displayed in Column 3 will only appear if the file system was mounted as a read-only or remote resource.

NAME

mvdir − move a directory

SYNOPSIS

/etc/mvdir dirname name

DESCRIPTION

The *mvdir* command moves directories within a file system. *Dirname* must be a directory. If *name* does not exist, it will be created as a directory. If *name* does exist, *dirname* will be created as *name/dirname*. *Dirname* and *name* may not be on the same path; that is, one may not be subordinate to the other. For example:

mvdir x/y x/z

is legal, but

mvdir x/y x/y/z

is not.

SEE ALSO

mkdir(1), mv(1).

WARNINGS

Only the super-user can use *mvdir*.

NAME

nawk — pattern scanning and processing language

SYNOPSIS

nawk [−**F** re] [parameter...] ['prog'] [−**f** progfile] [file...]

DESCRIPTION

nawk is a new version of *awk* that provides capabilities unavailable in previous versions. This version will become the default version of *awk* in the next major UNIX system release.

The −**F** *re* option defines the input field separator to be the regular expression *re.*

Parameters, in the form x=... y=... may be passed to *nawk,* where x and y are *nawk* built-in variables (see list below).

nawk scans each input *file* for lines that match any of a set of patterns specified in *prog*. The *prog* string must be enclosed in single quotes (') to protect it from the shell. For each pattern in *prog*, there may be an associated action performed when a line of a *file* matches the pattern. The set of pattern-action statements may appear literally as *prog* or in a file specified with the −**f** *progfile* option.

Input files are read in order; if there are no files, the standard input is read. The file name − means the standard input. Each input 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 normally made up of fields separated by white space. (This default can be changed by using the FS built-in variable or the −**F** *re* option.) The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

A pattern-action statement has the form:

    pattern { action }

Either pattern or action may be omitted. If there is no action with a pattern, the matching line is printed. If there is no pattern with an action, the action is performed on every input line.

Patterns are arbitrary Boolean combinations (**!**, **||**, **&&**, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

    expression relop expression
    expression matchop regular expression

where a relop is any of the six relational operators in C, and a matchop is either ˜ (contains) or !˜ (does not contain). A conditional is an arithmetic expression, a relational expression, the special expression

    var **in** array,

or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line has been read and after the last input line has been read, respectively.

Regular expressions are as in *egrep* [see *grep*(1)]. In patterns they must be surrounded by slashes. 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 pattern.

A regular expression may be used to separate fields by using the **−F** *re* option or by assigning the expression to the built-in variable FS. The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value, leading blanks are no longer ignored.

Other built-in variables include:

ARGC	command line argument count
ARGV	command line argument array
FILENAME	name of the current input file
FNR	ordinal number of the current record in the current file
FS	input field separator regular expression (default blank)
NF	number of fields in the current record
NR	ordinal number of the current record
OFMT	output format for numbers (default %.6g)
OFS	output field separator (default blank)
ORS	output record separator (default new-line)
RS	input record separator (default new-line)

An action is a sequence of statements. A statement may be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
do statement while ( conditional )
for ( expression ; conditional ; expression ) statement
for ( var in array ) statement
delete array[subscript]
break
continue
{ [ statement ] ... }
expression          # commonly variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next                # skip remaining patterns on this input line
exit [expr]         # skip the rest of the input; exit status is expr
return [expr]
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole input 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 or zero. 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 >*expression* is present, or on a pipe if | *cmd* is present. The arguments are 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) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*].

nawk has a variety of built-in functions: arithmetic, string, input/output, and general.

The arithmetic functions are: *atan2*, *cos*, *exp*, *int*, *log*, *rand*, *sin*, *sqrt*, and *srand*. *int* truncates its argument to an integer. *rand* returns a random number between 0 and 1. *srand* (expr) sets the seed value for *rand* to *expr* or uses the time of day if *expr* is omitted.

The string functions are:

gsub(*for*, *repl*, *in*)
: behaves like *sub* (see below), except that it replaces successive occurrences of the regular expression (like the *ed* global substitute command).

index (*s* , *t*) returns the position in string *s* where string *t* first occurs, or 0 if it does not occur at all.

lengthfl (*s*) returns the length of its argument taken as a string, or of the whole line if there is no argument.

match (*s* , *re*) returns the position in string *s* where the regular expression *re* occurs, or 0 if it does not occur at all. RSTART is set to the starting position (which is the same as the returned value), and RLENGTH is set to the length of the matched string.

split(*s*, *a*, *fs*) splits the string *s* into array elements *a*[*1*], *a*[*2*], *a*[*n*], and returns *n*. The separation is done with the regular expression *fs* or with the field separator FS if *fs* is not given.

sprintf(*fmt*, *expr*, *expr*, ...)
: formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

sub(*for*, *repl*, *in*)
: substitutes the string *repl* in place of the first instance of the regular expression *for* in string *in* and returns the number of substitutions. If *in* is omitted, *nawk* substitutes in the current record (**$0**).

substr(*s*, *m*, *n*) returns the *n*-character substring of *s* that begins at position *m*.

The input/output and general functions are:

close(*filename*) closes the file or pipe named *filename*.

cmd | *getline* pipes the output of *cmd* into *getline*; each successive call to *getline* returns the next line of output from *cmd*.

getline sets **$0** to the next input record from the current input file.

getline < *file* sets **$0** to the next record from *file*.

getline var sets variable *var* instead.

getline var < *file*
sets *var* from the next record of *file*.

system (*cmd*) executes *cmd* and returns its exit status.

All forms of *getline* return 1 for successful input, 0 for end of file, and −1 for an error.

nawk also provides user-defined functions. Such functions may be defined (in the pattern position of a pattern-action statement) as

```
function name(args,...) { stmts }
func name(args,...) { stmts }
```

Function arguments are passed by value if scalar and by reference if array name. Argument names are local to the function; all other variable names are global. Function calls may be nested and functions may be recursive. The **return** statement may be used to return a value.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { 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 }
```

Simulate *echo*(1):

```
BEGIN {
      for (i = 1; i < ARGC; i++)
            printf "%s", ARGV[i]
      printf "\n"
```

```
        exit
        }
```

Print file, filling in page numbers starting at 5:

```
/Page/  { $2 = n++; }
        { print }
```

command line: **nawk −f program n=5 input**

SEE ALSO

grep(1), sed(1).

lex(1), printf(3S) in the *INTERACTIVE SDS Guide and Programmer's Reference 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.

NAME

ncheck – generate path names from i-numbers

SYNOPSIS

/etc/ncheck [**–i** i-numbers] [**–a**] [**–s**] [file-system]

DESCRIPTION

The *ncheck* command with no arguments generates a path name vs. i-number list of all files on a set of default file systems (see */etc/checklist*). Names of directory files are followed by /..

The options are as follows:

–i limits the report to only those files whose i-numbers follow.

–a allows printing of the names . and .., which are ordinarily suppressed.

–s limits the report to special files and files with set-user-ID mode. This option may be used to detect violations of security policy.

File system must be specified by the file system's special file.

The report should be sorted so that it is more useful.

SEE ALSO

fsck(1M), sort(1).

DIAGNOSTICS

If the file system structure is not consistent, ?? denotes the "parent" of a parentless file, and a path name beginning with ... denotes a loop.

NAME

newaliases − rebuild the database for the mail aliases file

SYNOPSIS

newaliases

DESCRIPTION

The *newaliases* program rebuilds the random access database for the mail aliases file **/usr/lib/aliases**. It must be run each time **/usr/lib/aliases** is changed in order for the change to take effect.

SEE ALSO

sendmail(8).
aliases(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

newform − change the format of a text file

SYNOPSIS

newform [−s] [−itabspec] [−otabspec] [−bn] [−en] [−pn] [−an] [−f] [−cchar] [−ln] [files]

DESCRIPTION

The *newform* command 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 "**−e**15 **−l**60" will yield results different from "**−l**60 **−e**15". Options are applied to all *files* on the command line.

−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

−i*tabspec* 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**.

−o*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **−i***tabspec*. If no *tabspec* is given, *tabspec* defaults to **−8**. A *tabspec* of **−0** means that no spaces will be converted to tabs on output.

−b*n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **−l***n*). 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 −l1 −b7 file-name

−e*n* Same as **−b***n* except that characters are truncated from the end of the line.

−p*n* Prefix *n* characters (see **−c***k*) 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.

−a*n* Same as **−p***n* 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**.

−c*k* Change the prefix/append character to *k*. Default character for *k* is a space.

−l*n* 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).

The **−l1** must be used to set the effective line length shorter than any existing line in the file so that the **−b** option is activated.

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*newform* was called with a bad option.
not −s format	There was no tab on one line.
can't open file	Self-explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

0 − normal execution
1 − for any error

SEE ALSO

csplit(1), tabs(1).

fspec(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

BUGS

The *newform* command 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.

NAME

newgrp – log in to a new group

SYNOPSIS

newgrp [–] [group]

DESCRIPTION

The *newgrp* command 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. This is a way to exit the effect of an earlier *newgrp* command.

If the first argument to *newgrp* is a –, the environment is changed to what would be expected if the user actually logged in again as a member of the new group.

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

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), sh(1).
group(4), passwd(4), environ(5) in the *INTERACTIVE SDS Guide and Programmer's Reference 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

The *news* command 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."

–a option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

–n option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

–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 *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

nice − run a command at low priority

SYNOPSIS

nice [−increment] command [arguments]

DESCRIPTION

The *nice* command 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 *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

DIAGNOSTICS

The *nice* command 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

The *nl* command 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):

Line contents	*Start of*
\:\:\:	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:

−b*type* 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
- **p***string* 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 **−b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

−f*type* Same as **−b***type* except for footer. Default for logical page footer is **n** (no lines numbered).

−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**.

−p Do not restart numbering at logical page delimiters.

−l*num* *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**.

−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).

−d*xx* 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

nlsadmin – network listener service administration

SYNOPSIS

nlsadmin –x
nlsadmin [options] net_spec

DESCRIPTION

nlsadmin administers the network listener process(es) on a machine. Each network has a separate instance of the network listener process associated with it; each instance (and thus, each network) is configured separately. The listener process "listens" to the network for service requests, accepts requests when they arrive, and spawns servers in response to those service requests. The network listener process will work with any network (more precisely, with any transport provider) that conforms to the transport provider specification.

The listener supports two classes of service: a general listener service, serving processes on remote machines, and a terminal login service, for terminals connected directly to a network. The terminal login service provides networked access to this machine in a form suitable for terminals connected directly to the network. However, this direct terminal service requires special associated software, and is only available with some networks (for example, the AT&T STARLAN network).

nlsadmin can establish a listener process for a given network, configure the specific attributes of that listener, and start and kill the listener process for that network. *nlsadmin* can also report on the listener processes on a machine, either individually (per network) or collectively.

The following list shows how to use *nlsadmin*. In this list, *net_spec* represents a particular listener process. Specifically, *net_spec* is the relative path name of the entry under /**dev** for a given network (that is, a transport provider). Changing the list of services provided by the listener produces immediate changes, while changing an address on which the listener listens has no effect until the listener is restarted. The following combination of *options* can be used.

no options — will give a brief usage message.

–x — will report the status of all of the listener processes installed on this machine.

net_spec — will print the status of the listener process for *net_spec*.

–q *net_spec* — will query the status of the listener process for the specified network, and will reflect the result of that query in its exit code. If a listener process is active, *nlsadmin* will exit with a status of 0; if no process is active, the exit code will be 1; the exit code will be greater than 1 in case of error.

–v *net_spec* — will print a verbose report on the servers associated with *net_spec*, giving the service code, status, command, and comment for each. It also specifies the **uid** the server will run as, and the list of modules to be pushed, if any, before the server is started.

−z *service_code net_spec*
will print a report on the server associated with *net_spec* that has service code *service_code*, giving the same information as in the **−v** option.

−q **−z** *service_code net_spec*
will query the status of the service with service code *service_code* on network *net_spec*, and will exit with a status of 0 if that service is enabled, 1 if that service is disabled, and greater than 1 in case of error.

−l *addr net_spec*
will change or set the address on which the listener listens (the general listener service). This is the address generally used by remote processes to access the servers available through this listener (see the **−a** option, below). *addr* is the transport address on which to listen and is interpreted using a syntax that allows for a variety of address formats. By default *addr* is interpreted as the symbolic ASCII representation of the transport address. An *addr* preceded by a **\x** will let you enter an address in hexadecimal notation. Note that *addr* must appear as a single word to the shell and must be quoted if it contains any blanks.

If *addr* is just a dash ("−"), *nlsadmin* will report the address currently configured, instead of changing it.

A change of address will not take effect until the next time the listener for that network is started.

−t *addr net_spec*
will change or set the address on which the listener listens for requests for terminal service, but is otherwise similar to the **−l** option above. A terminal service address should not be defined unless the appropriate remote login software is available; if such software is available, it must be configured as service code 1 (see the **−a** option, below).

−i *net_spec*
will initialize or change a listener process for the network specified by *net_spec*; that is, it will create and initialize the files required by the listener. Note that the listener should only be initialized once for a given network, and that doing so does not actually invoke the listener for that network. The listener must be initialized before assigning addressing or services.

[−m] −a *service_code* **[−p** *modules***] [−w** *id***] −c** *cmd* **−y** *comment net_spec*
will add a new service to the list of services available through the indicated listener. *service_code*

is the code for the service, *cmd* is the command to be invoked in response to that service code, comprised of the full path name of the server and its arguments, and *comment* is a brief (free-form) description of the service for use in various reports. Note that *cmd* must appear as a single word to the shell, so if arguments are required, the *cmd* and its arguments must be surrounded by quotes. Similarly, the *comment* must also appear as a single word to the shell. When a service is added, it is initially enabled (see the **−e** and **−d** options below).

If the **−m** option is specified, the entry will be marked as an administrative entry. Service codes 1 through 100 are reserved for administrative entries, which are those that require special handling internally. In particular, code 1 is assigned to the remote login service, which is the service automatically invoked for connections to the terminal login address.

The **−m** option used with the **−a** option indicates that special handling internally is required for those servers added with the **−m** set. This internal handling is in the form of code embedded on the listener process.

If the **−p** option is specified, then *modules* will be interpreted as a list of STREAMS modules for the listener to push before starting the service being added. The modules are pushed in the order they are specified. *modules* should be a comma-separated list of modules, with no white space included.

If the **−w** option is specified, then *id* is interpreted as the user name from **/etc/passwd** that the listener should look up. From the user name, the listener should obtain the user ID, the group ID, and the home directory for use by the server. If **−w** is not specified, the default is to use the user ID **listen**.

A service must explicitly be added to the listener for each network on which that service is to be available. This operation will normally be performed only when the service is installed on a machine, or when populating the list of services for a new network.

−r *service_code net_spec*
will remove the entry for the *service_code* from that listener's list of services. This will normally be performed only in conjunction with the de-installation of a service from a machine.

−e *service_code ne spec*
−d *service_code net_spec*
will enable or disable (respectively) the service indicated by *service_code* for the specified network. The service must have previously been added to the listener for that network (see the **−a** option above). Disabling a service will cause subsequent service requests for that service to be denied, but the processes from any prior service requests that are still running will continue unaffected.

−s *net_spec*
−k *net_spec* will start and kill (respectively) the listener process for the indicated network. These operations will normally be performed as part of the system startup and shutdown procedures. Before a listener can be started for a particular network, it must first have been initialized, and an address must be defined for the general listener service (see the **−i** and **−l** options, above). When a listener is killed, processes that are still running as a result of prior service requests will continue unaffected.

The listener runs as user ID **root**, with group ID **sys**. A special ID, user ID **listen** and group ID **adm**, should be entered in the **/etc/passwd** file as a default ID for servers. The listener always uses as its home directory **/usr/net/nls**, which is concatenated with *net_spec* to determine the location of the listener configuration information for each network. The home directory specified in the **/etc/passwd** entry for **listener** will used by servers that run as ID **listen**.

nlsadmin may be invoked by any user to generate reports, but all operations that affect a listener's status or configuration are restricted to the super-user.

FILES

/usr/net/nls/*net_spec*

SEE ALSO

Network Programmer's Guide.

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

nohup command [arguments]

DESCRIPTION

The *nohup* command 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:

```
sort ofile > nfile
```

SEE ALSO

chmod(1), nice(1), sh(1),
signal(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

WARNINGS

In the case of the following command

```
nohup command1; command2
```

nohup applies only to command1. The command

```
nohup (command1; command2)
```

is syntactically incorrect.

NAME

od – octal dump

SYNOPSIS

od [–bcdosx] [file **] [[+**]offset[**.**][**b**]]

DESCRIPTION

The *od* command 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 non-graphic 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.

NAME

pack, pcat, unpack — compress and expand files

SYNOPSIS

pack [**—**] [**—f**] name ...

pcat name ...

unpack name ...

DESCRIPTION

The *pack* command 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*.

The *pack* command 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.

The *pack* command 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.

The *pcat* command 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

The *pcat* command 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).

NAME

passmgmt – password files management

SYNOPSIS

passmgmt −a options name
passmgmt −m options name
passmgmt −d name

DESCRIPTION

The *passmagmt* command updates information in the password files. This command works with both **/etc/passwd** and **/etc/shadow**. If there is no shadow password file the changes done by *passmgmt* will go in **/etc/passwd.**

passmgmt −a adds an entry for user *name* to the login password files. This command does not create any directory for the new user and the new login remains locked (with the string LK in the *password* field) until the *passwd*(1M) command is executed to set the password.

passmgmt −m modifies the entry for user *name* in the login password files. The name field in the **/etc/shadow** entry and all the fields (except the password field) in the **/etc/password** entry can be modified by this command. Only fields entered on the command line will be modified. If there is no **/etc/shadow** file, all modifications are made in **/etc/passwd.**

passmgmt −d deletes the entry for user *name* from the login password files. It will not remove any files that the user owns on the system; they must be removed manually.

name, the login name of the user, must be unique.

The following options are available:

−c *comment* A short description of the login. It is limited to a maximum of 128 characters and defaults to an empty field.

−h *homedir* Home directory of *name*. It is limited to a maximum of 256 characters and defaults to **/usr/***name*.

−u *uid* UID of the *name*. This number must range from 0 to the maximum value for the system. It defaults to the next available UID greater than 100. Without the **−o** option, it enforces the uniqueness of a UID.

−o This option allows a UID to be non-unique. It is used only with the **−u** option.

−g *gid* GID of the *name*. This number must range from 0 to the maximum value for the system. The default is 1.

−s *shell* Login shell for *name*. It should be the full pathname of the program that will be executed when the user logs in. The maximum length of *shell* is 256 characters. The default is for this field to be empty and to be interpreted as **/bin/sh.**

−l *logname* This option changes the *name* to *logname* for the **−m** option only.

The total size of each login entry, whether existing or new, is limited to a maximum of 511 bytes in the password files.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

passwd(1), passwd(1M).

passwd(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

DIAGNOSTICS

The *passmgmt* command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid command syntax. Usage message of the *passmgmt* command will be displayed.
3	Invalid argument provided to option.
4	UID in use.
5	Inconsistent password files (e.g., *name* is in the **/etc/passwd** file and not in the **/etc/shadow** file, or vice versa).
6	Unexpected failure. Password files unchanged.
7	Unexpected failure. Password file(s) missing.
8	Password file(s) busy. Try again later.
9	*name* does not exist (if **−m** or **−d** is specified), already exists (if **−a** is specified), or *logname* already exists (if **−m −l** is specified).

NOTE

You cannot use a colon or <cr> because it will be interpreted as a field separator.

NAME

passwd – change login password and password attributes

SYNOPSIS

passwd [name]

passwd **–s** [name]

passwd **–l** [**–f**] [**–x** max] [**–n** min] name

passwd **–d** [**–f**] [**–x** max] [**–n** min] name

passwd **–s** [**–a**]

DESCRIPTION

The *passwd* command changes the password or lists password attributes associated with the user's login *name*. Additionally, super-users may use *passwd* to install or change passwords and attributes associated with any login *name*. (Options relating to password attributes are only available on systems using the **/etc/shadow** security feature.)

When used to change a password, *passwd* prompts ordinary users for their old password, if any. It then prompts for the new password twice. When the old password is entered, *passwd* checks to see if it has "aged" sufficiently. If "aging" is insufficient, *passwd* terminates; see *passwd*(4).

If the user's password aging has not been turned on, then password aging is turned on for the user using the **MAXWEEK** and **MINWEEK** parameters in **/etc/default/passwd**. If password aging is turned on, the password aging information in **/etc/shadow** remains unmodified.

Assuming aging is sufficient, a check is made to ensure 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 **PASSLENGTH** characters as set in **/etc/default/passwd**. **PASSLENGTH** must contain a minimum of three characters, but 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" refers to all uppercase or 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.

Super-users [e.g., real and effective uid equal to zero, see *id*(1M) and *su*(1M)] 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. (This differs from *passwd* **−d** because the "password" prompt will still be displayed.)

Any user may use the **−s** option to show password attributes for the login *name*.

The format of the display will be

name status mm/dd/yy min max

or, if password aging information is not present,

name status

where:

name The login ID of the user.

status The password status of *name*: PS stands for passworded or locked, LK stands for locked, and NP stands for no password.

mm/dd/yy The date password was last changed for *name*.

min The minimum number of days required between password changes for *name*.

max The maximum number of days the password is valid for *name*.

Only a super-user can use the following options:

−l Locks password entry for *name*.

−d Deletes password for *name*. The login *name* will not be prompted for password.

−n Set minimum field for *name*. The *min* field contains the minimum number of days between password changes for *name*. Always use this option with the **−x** option (except when **−x** *man* is set to −1) to insure that aging is turned on.

−x Set maximum field for *name*. The *max* field contains the number of days that the password is valid for *name*. The aging for *name* will be turned off immediately if *max* is set to −1. (Do not use with the **−n** option.) If it is set to 0, then the user is forced to change the password and aging is turned off at the next day's login session.

−a Show password attributes for all entries. Use only with **−s** option; *name* must not be provided.

−f Force the user to change password at the next login by expiring the password for *name*.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow
/etc/default/passwd

SEE ALSO

login(1), id(1M), passmgmt(1M), pwconv(1M), su(1M).

crypt(3C), passwd(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

DIAGNOSTICS

The *passwd* command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid combination of options.
3	Unexpected failure. Password file unchanged.
4	Unexpected failure. Password file(s) missing.
5	Password file(s) busy. Try again later.
6	Invalid argument to option.

WARNING

If root deletes a password for a user with the *passwd −d* command, and password aging is in effect for that user, the user will not be allowed to add a new password until the NULL password has been aged. This is true even if the PASSREQ flag in **/etc/login/default** is set to YES. This results in a user without a password. It is recommended that the **−f** option be used whenever the **−d** (delete) option is used. This will force a user to change the password at next login.

NAME

passwd – change login password and password attributes

SYNOPSIS

passwd [name]

passwd –l [**–f**] [**–x** max] [**–n** min] name

passwd –d [**–f**] [**–x** max] [**–n** min] name

passwd –s [**–a**]

passwd –s [name]

DESCRIPTION

This command changes or installs login passwords and password attributes associated with the login *name*. The options to *passwd* are:

–l Locks password entry for *name*.

–d Deletes password for *name*. The login *name* will not be prompted for password.

–x Set maximum field for *name*. The *max* field contains the number of days that the password is valid for *name*. The aging for *name* will be turned off if *max* is set to 0.

–n Set minimum field for *name*. The *min* field contains the minimum number of days between password changes for *name*.

–s Shows password attributes for *name*. The format of the display will be:

name status mm/dd/yy min max

or, if password aging information is not present,

name status

name is the login ID of the user. *status* is the password status of *name*. PS stands for passworded or locked, LK stands for locked, and NP stands for no password. *mm/dd/yy* is the date password was last changed for *name*. *min* is the minimum number of days required between password changes for *name*. *max* is the maximum number of days the password is valid for *name*.

–a Show password attributes for all entries. Use only with **–s** option; *name* must not be provided.

–f Force the user to change password at the next login by expiring the password for *name*.

Privileged users may change any password; hence, *passwd* does not prompt privileged users for the old password. Privileged users are not forced to comply with password aging and password construction requirements. A privileged user can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow
/etc/default/passwd

SEE ALSO

id(1M), login(1), passmgmt(1M), passwd(1), su(1M).

crypt(3C), passwd(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

DIAGNOSTICS

The *passwd* command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid combination of options.
3	Unexpected failure. Password file unchanged.
4	Unexpected failure. Password file missing.
5	Password file(s) busy. Try again later.
6	Invalid argument to option.

NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste –d list file1 file2 ...
paste –s [**–d** list] 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

pax – portable archive exchange

SYNOPSIS

pax [–cimopuvy] [–f *archive***] [–s** *replstr***] [–t** *device***]** [*pattern...*]

pax –r [–cimnopuvy] [–f *archive***] [–s** *replstr***] [–t** *device***]** [*pattern...*]

pax –w [–adimuvy] [–b *blocking***] [–f** *archive***] [–s** *replstr***] [–t** *device***] [–x** *format***]** [*pathname...*]

pax –rw [–ilmopuvy] [–s *replstr***]** [*pathname...*] directory

DESCRIPTION

The *pax* utility reads and writes archive files that conform to the "Archive/Interchange File Format" specified in *IEEE Std. 1003.1-1988*. The *pax* utility can also read, but not write, a number of other file formats in addition to those specified in the "Archive/Interchange File Format" description. Support for these traditional file formats, such as V7 *tar* and System V binary *cpio* format archives, is provided for backward compatibility and to maximize portability.

Combinations of the **–r** and **–w** command line arguments specify whether *pax* will read, write, or list the contents of the specified archive, or move the specified files to another directory.

The command line arguments are:

–w This argument writes the files and directories specified by *pathname* operands to the standard output together with the path name and status information prescribed by the archive format used. A directory *pathname* operand refers to the files and (recursively) subdirectories of that directory. If no *pathname* operands are given, then the standard input is read to get a list of path names to copy, one path name per line. In this case, only those path names appearing on the standard input are copied.

–r The *pax* utility reads an archive file from the standard input. Only files with names that match any of the *pattern* operands are selected for extraction. The selected files are conditionally created and copied relative to the current directory tree, subject to the options described below. By default, the owner and group of selected files will be that of the invoking process, and the permissions and modification times will be the same as those in the archive.

The supported archive formats are automatically detected on input. The default output format is *ustar*, but that may be overridden by the **–x** *format* option described below.

–rw The *pax* utility reads the files and directories named in the *pathname* operands and copies them to the destination *directory*. A directory *pathname* operand refers to the files and (recursively) subdirectories of that directory. If no *pathname* operands are given, the standard input is read to get a list of path names to copy, one path name per line. In this case, only those path names appearing on the standard input are copied.

The directory named by the *directory* operand must exist and it must have the proper permissions before the copy can occur.

If neither the **−r** or **−w** options are given, *pax* will list the contents of the specified archive. In this mode, *pax* lists normal files one per line, showing link path names as:

pathname == linkname

If the **−v** option is specified, then *pax* lists normal path names in the same format used by the *ls* utility with the **−l** option. Links are shown as:

<ls -l listing> == linkname

The *pax* utility is capable of reading and writing archives that span multiple physical volumes. Upon detecting an end of medium on an archive that is not yet completed, *pax* will prompt the user for the next volume of the archive and will allow the user to specify the location of the next volume.

Options

The following options are available:

−a The files specified by *pathname* are appended to the specified archive.

−b *blocking* Block the output at *blocking* bytes per write to the archive file. A **k** suffix multiplies *blocking* by 1024, a **b** suffix multiplies *blocking* by 512, and an **m** suffix multiplies *blocking* by 1048576 (1 megabyte). If not specified, *blocking* is automatically determined on input and is ignored for **−rw**.

−c Complement the match sense of the *pattern* operands.

−d Intermediate directories not explicitly listed in the archive are not created. This option is ignored unless the **−r** option is specified.

−f *archive* The *archive* option specifies the path name of the input or output archive, overriding the default of standard input for **−r** or standard output for **−w**.

−i Interactively rename files. Substitutions specified by **−s** options (described below) are performed before requesting the new file name from the user. A file is skipped if an empty line is entered, and *pax* exits with an exit status of 0 if **EOF** is encountered.

−l Files are linked rather than copied when possible.

−m File modification times are not retained.

−n When **−r** is specified, but **−w** is not, the *pattern* arguments are treated as ordinary file names. Only the first occurrence of each of these files in the input archive is read. The *pax* utility exits with a zero exit status after all files in the list have been read. If one or more files in the list is not found, *pax* writes a diagnostic to standard error for each of the files and exits with a non-zero exit status. The file names are compared before any of the **−i**, **−s**, or **−y** options are applied.

−o Restore file ownership as specified in the archive. The invoking process must have appropriate privileges to accomplish this.

−p Preserve the access time of the input files after they have been copied.

−s *replstr* Filenames are modified according to the substitution expression using the syntax of *ed*(1) as shown:

−s */old/new/*[gp]

Any non-null character may be used as a delimiter (a / is used in this example). Multiple **−s** expressions may be specified; the expressions are applied in the order specified terminating with the first successful substitution. The optional trailing **p** causes successful mappings to be listed on standard error. The optional trailing **g** causes the *old* expression to be replaced each time it occurs in the source string. Files that substitute to an empty string are ignored both on input and output.

−t *device* The *device* option argument is an implementation-defined identifier that names the input or output archive device, overriding the default of standard input for **−r** and standard output for **−w**.

−u Copy each file only if it is newer than a pre-existing file with the same name. This implies **−a**.

−v List file names as they are encountered. Produces a verbose table of contents listing on the standard output when both **−r** and **−w** are omitted; otherwise the file names are printed to standard error as they are encountered in the archive.

−x *format* Specifies the output archive *format*. The input format, which must be one of the following, is automatically determined when the **−r** option is used. The supported formats are:

cpio The extended *CPIO* interchange format specified in "Extended CPIO Format" in *IEEE Std. 1003.1-1988*.

ustar The extended *TAR* interchange format specified in "Extended TAR Format" in *IEEE Std. 1003.1-1988*. This is the default archive format.

−y Interactively prompt for the disposition of each file. Substitutions specified by **−s** options (described above) are performed before prompting the user for disposition. **EOF** or an input line starting with the character **q** cause *pax* to exit. Otherwise, an input line starting with anything other than **y** causes the file to be ignored. This option cannot be used in conjunction with the **−i** option.

Only the last of multiple **−f** or **−t** options take effect.

When writing to an archive, the standard input is used as a list of path names if no *pathname* operands are specified. The format is one path name per line. Otherwise, the standard input is the archive file.

The user ID and group ID of the process, together with the appropriate privileges, affect the ability of *pax* to restore ownership and permissions attributes of the archived files.

The options **−a**, **−c**, **−d**, **−i**, **−l**, **−p**, **−t**, **−u**, and **−y** are provided for functional compatibility with the historical *cpio* and *tar* utilities. The option defaults were chosen based on the most common usage of these options; therefore, some of the options have meanings different than those of the historical commands.

Operands

The following operands are available:

directory — The destination directory path name for copies when both the **−r** and **−w** options are specified. The directory must exist and be writable before the copy or error results.

pathname — A file whose contents are used instead of the files named on the standard input. When a directory is named, all of its files and (recursively) subdirectories are copied as well.

pattern — A *pattern* is given in the standard shell pattern matching notation. The default if no *pattern* is specified is *, which selects all files.

EXAMPLES

The following command:

```
pax −w −f /dev/rmt0 .
```

copies the contents of the current directory to tape drive 0.

The commands

mkdir *newdir*
cd *olddir*
pax -rw . ../*newdir*

copies the contents of *olddir* to *newdir*.

The command

```
pax −r −s ',//*usr//*,,' -f pax.out
```

reads the archive **pax.out** with all files rooted in **/usr** in the archive extracted relative to the current directory.

FILES

/dev/tty — Used to prompt the user for information when the **−i** or **−y** options are specified

SEE ALSO

cpio(1), find(1), tar(1).
cpio(5), tar(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

DIAGNOSTICS

pax will terminate immediately, without processing any additional files on the command line or in the archive.

EXIT CODES

pax will exit with one of the following values:

0 All files in the archive were processed successfully.

>0 *pax* aborted due to errors encountered during operation.

BUGS

Special permissions may be required to copy or extract special files.

Device, user ID, and group ID numbers larger than 65535 cause additional header records to be output. These records are ignored by some historical versions of *cpio*(1) and *tar*(1).

The archive formats described in "Archive/Interchange File Format" have certain restrictions that have been carried over from historical usage. For example, there are restrictions on the length of path names stored in the archive.

When getting an *ls* **–l** style listing on *tar* format archives, link counts are listed as zero since the *ustar* archive format does not keep link count information.

SOURCE

This version of the *pax* utility is derived from software developed by Mark H. Colburn and sponsored by The USENIX Association.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

pg – file perusal filter for CRTs

SYNOPSIS

pg [– number] [**–p** string] [**–cefns**] [+ linenumber]
[+/ pattern /] [files ...]

DESCRIPTION

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a CRT. (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) **l**
: 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 backwards 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:

***i*n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

***i*p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

***i*w** 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.

h 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

/usr/lib/terminfo/?/*	terminal information database
/tmp/pg*	temporary file when input is from a pipe

SEE ALSO

ed(1), grep(1).
terminfo(4) in the *INTERACTIVE SDS Guide and Programmer's Reference 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, terminal settings may not be restored correctly.

NAME

pr − print files

SYNOPSIS

pr [[**−column**] [**−w**width] [**−a**]] [**−e**ck] [**−i**ck] [**−drtfp**] [+page] [**−n**ck] [**−o**offset] [**−l**length] [**−s**separator] [**−h**header] [file ...]

pr [[**−m**] [**−w**width]] [**−e**ck] [**−i**ck] [**−drtfp**] [+page] [**−n**ck] [**−o**offset] [**−l**length] [**−s**separator] [**−h**header] file1 file2 ...

DESCRIPTION

pr is used to format and print the contents of a file. If *file* is **−**, or if no files are specified, *pr* assumes standard input. *pr* prints the named files on standard output.

By default, the listing is separated into pages, each headed by the page number, the date and time that the file was last modified, and the name of the file. Page length is 66 lines, which includes 10 lines of header and trailer output. The header is composed of 2 blank lines, 1 line of text (can be altered with **−h**), and 2 blank lines; the trailer is 5 blank lines. For single column output, line width may not be set and is unlimited. For multi-column output, line width may be set and the default is 72 columns. Diagnostic reports (failed options) are reported at the end of standard output associated with a terminal, rather than interspersed in the output. Pages are separated by series of line feeds rather than form feed characters.

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 *separator* character.

Either *−column* or **−m** should be used to produce multi-column output. **−a** should only be used with *−column* and not **−m**.

Command line options are:

+page Begin printing with page numbered *page* (default is 1).

−column Print *column* columns of output (default is 1). Output appears as if **−e** and **−i** are turned on for multi-column output. May not use with **−m**.

−a Print multi-column output across the page one line per column. *columns* must be greater than one. If a line is too long to fit in a column, it is truncated.

−m Merge and print all files simultaneously, one per column. The maximum number of files that may be specified is eight. If a line is too long to fit in a column, it is truncated. May not use with *−column.*

−d Double-space the output. Blank lines that result from double-spacing are dropped when they occur at the top of a page.

−e*ck* Expand input tabs to character positions $k+1$, $2*k+1$, $3*k+1$, 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).

−i*ck* In output, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

−n*ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of single column 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).

−w*width* Set the width of a line to *width* character positions (default is 72). This is effective only for multi-column output (-*column* and **−m**). There is no line limit for single column output.

−o*offset* Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

−l*length* Set the length of a page to *length* lines (default is 66). **−l0** is reset to **−l66**. When the value of *length* is 10 or less, **−t** appears to be in effect since headers and trailers are suppressed. By default, output contains 5 lines of header and 5 lines of trailer leaving 56 lines for user-supplied text. When **−l***length* is used and *length* exceeds 10, then *length*-10 lines are left per page for user-supplied text. When *length* is 10 or less, header and trailer output is omitted to make room for user-supplied text.

−h *header* Use *header* as the text line of the header to be printed instead of the file name. **−h** is ignored when **−t** is specified or **−l***length* is specified and the value of *length* is 10 or less. (**−h** is the only *pr* option requiring space between the option and argument.)

−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).

−f Use single 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 files that will not open.

−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. Use of **−t** overrides the **−h** option.

−s*separator*
Separate columns by the single character *separator* instead of by the appropriate number of spaces (default for *separator* is a tab). Prevents truncation of lines on multi-column output unless **−w** is specified.

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr −3dh "file list" file1 file2
```

Copy **file1** to **file2**, expanding tabs to columns 10, 19, 28, 37, ... :

```
pr −e9 −t <file1 > file2
```

Print **file1** and **file2** simultaneously in a two-column listing with no header or trailer where both columns have line numbers:

```
pr −t −n file1 | pr −t −m −n file2
```

FILES

/dev/tty* If standard output is directed to one of the special files **/dev/tty***, then other output directed to this terminal is delayed until standard output is completed. This prevents error messages from being interspersed throughout the output.

SEE ALSO

cat(1), pg(1).

NAME

profiler: prfld, prfstat, prfdc, prfsnap, prfpr — UNIX system profiler

SYNOPSIS

/etc/prfld [system_namelist]
/etc/prfstat on
/etc/prfstat off
/etc/prfdc file [period [off_hour]]
/etc/prfsnap file
/etc/prfpr file [cutoff [system_namelist]]

DESCRIPTION

The *prfld*, *prfstat*, *prfdc*, *prfsnap*, and *prfpr* routines form a system of programs to facilitate an activity study of the UNIX operating system.

The *prfld* program 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 *system_namelist*.

The *prfstat* program 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.

The *prfdc* and *prfsnap* programs 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*.

The *prfpr* program formats the data collected by *prfdc* or *prfsnap*. Each text address is converted to the nearest text symbol (as found in *system_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 system namelist file

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 controlling terminal. 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 accept names or lists as arguments. Arguments can be either separated from one another by commas or enclosed in double quotes and separated from one another by commas or spaces. Values for *proclist* and *grplist* must be numeric.

The *options* are given in descending order according to volume and range of information provided:

—e Print information about every process now running.

—d Print information about all processes except process group leaders.

—a Print information about all processes most frequently requested: all those except process group leaders and processes not associated with a terminal.

—f Generate a full listing. (See below for significance of columns in a full listing.)

—l Generate a long listing. (See the following text.)

—n *name* Valid only for users with a real user id of **root** or a real group id of **sys**. Takes argument signifying an alternate system *name* in place of **/unix**.

—t *termlist* List only process data associated with the terminal 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* List only process data whose process ID numbers are given in *proclist.*

—u *uidlist* List only process data whose user ID number or login name is given in *uidlist*. In the listing, the numerical user ID will be printed unless you give the **—f** option, which prints the login name.

—g *grplist* List only process data whose process group leader's ID number(s) appears in *grplist.* (A group leader is a process whose process ID number is identical to its process group ID number. A login shell is a common example of a process group leader.)

Under the **—f** option, *ps* tries to determine the command name and arguments given when the process was created by examining the user block. Failing this, the command name is printed, as it would have appeared without the **—f** option, in square brackets.

The column headings and the meaning of the columns in a *ps* listing are given in the following text; the letters **f** and **l** indicate the option (**f**ull or **l**ong, respectively) 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 (l) Flags (hexadecimal and additive) associated with the process

- 00 Process has terminated: process table entry now available.
- 01 A system process: always in primary memory.
- 02 Parent is tracing process.
- 04 Tracing parent's signal has stopped process: parent is waiting [*ptrace*(2)].
- 08 Process is currently in primary memory.
- 10 Process currently in primary memory: locked until an event completes.

S (l) The state of the process:

- O Process is running on a processor.
- S Sleeping: process is waiting for an event to complete.
- R Runnable: process is on run queue.
- I Idle: process is being created.
- Z Zombie state: process terminated and parent not waiting.
- T Traced: process stopped by a signal because parent is tracing it.
- X SXBRK state: process is waiting for more primary memory.

UID (f,l) The user ID number of the process owner (the login name is printed under the **−f** option).

PID (all) The process ID of the process (this datum is necessary in order to kill a process).

PPID (f,l) The process ID of the parent process.

C (f,l) Processor utilization for scheduling.

PRI (l) The priority of the process (higher numbers mean lower priority).

NI (l) Nice value, used in priority computation.

ADDR (l) The physical memory address of the first page of the user block. If the user block is swapped out, **ADDR** is shown as **0**.

SZ (l) The size (in pages or clicks) of the swappable process's image in main memory.

WCHAN	(l)	The address of an event for which the process is sleeping, or in SXBRK state, (if blank, the process is running).
STIME	(f)	The starting time of the process, given in hours, minutes, and seconds. (A process begun more than twenty-four hours before the *ps* inquiry is executed is given in months and days.)
TTY	(all)	The controlling terminal for the process (the message, ?, is printed when there is no controlling terminal).
TIME	(all)	The cumulative execution time for the process.
COMMAND	(all)	The command name (the full command name and its arguments are printed under the **−f** option).

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

FILES

/dev	
/dev/sxt/*	
/dev/tty*	
/dev/xt/*	terminal ("tty") names searcher files
/dev/kmem	kernel virtual memory
/dev/swap	the default swap device
/dev/mem	memory
/etc/passwd	UID information supplier
/etc/ps_data	internal data structure
/unix	system name list

SEE ALSO

getty(1M), kill(1), nice(1).

WARNING

Things can change while *ps* is running; the snap-shot it gives is only true for a split-second, and it may not be accurate by the time you see it. Some data printed for defunct processes is irrelevant.

If no *termlist*, *proclist*, *uidlist*, or *grplist* is specified, *ps* checks *stdin*, *stdout*, and *stderr* in that order, looking for the controlling terminal and will attempt to report on processes associated with the controlling terminal. In this situation, if *stdin*, *stdout*, and *stderr* are all redirected, *ps* will not find a controlling terminal, so there will be no report.

On a heavily loaded system, *ps* may report an *lseek*(2) error and exit. *ps* may seek to an invalid user area address: having obtained the address of a process' user area, *ps* may not be able to seek to that address before the process exits and the address becomes invalid.

ps −ef may not report the actual start of a tty login session, but rather an earlier time, when a getty was last respawned on the tty line.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

pwck, grpck – password/group file checkers

SYNOPSIS

/etc/pwck [file]
/etc/grpck [file]

DESCRIPTION

The *pwck* command 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 the program-to-use-as-Shell exist. The default password file is **/etc/passwd**.

The *grpck* command 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 *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

NAME

pwconv – install and update /etc/shadow with information from /etc/passwd

SYNOPSIS

pwconv

DESCRIPTION

The *pwconv* command creates and updates **/etc/shadow** with information from **/etc/passwd**. If the **/etc/shadow** file does not exist, this command will create **/etc/shadow** with information from **/etc/passwd**. The command populates **/etc/shadow** with the user's login name, password, and password aging information. If password aging information does not exist in **/etc/passwd** for a given user, none will be added to **/etc/shadow**. However, the "last changed" information will always be updated.

If the **/etc/shadow** file does exist, the following tasks will be performed:

> Entries that are in the **/etc/passwd** file and not in the **/etc/shadow** file will be added to the **/etc/shadow** file.
>
> Entries that are in the **/etc/shadow** file and not in the **/etc/passwd** file will be removed from **/etc/shadow**.
>
> Password attributes (e.g., password and aging information) that exist in an **/etc/passwd** entry will be moved to the corresponding entry in **/etc/shadow**.

The following is the format of an entry in **/etc/passwd**:

name:passwd,aging:uid:gid:comment:homedir:shell

The following table shows how changes made to **/etc/passwd** affect **/etc/shadow** when *pwconv* is run:

name	delete old entry from **/etc/shadow**, add new entry to **/etc/shadow**
passwd	update **password** and **aging** fields in **/etc/shadow**, place an *x* in **/etc/passwd**
aging	update **aging** fields in **/etc/shadow**, clear **aging** field in **/etc/passwd**
remaining fields	ignore—no change to **/etc/shadow**

The *pwconv* program is a privileged system command that cannot be executed by ordinary users. The *passwd* command should be used to add or change password aging information or passwords.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

passwd(1M), passmgmt(1M).

DIAGNOSTICS

The *pwconv* command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid command syntax.
3	Unexpected failure. Conversion not done.
4	Unexpected failure. Password file(s) missing.
5	Password file(s) busy. Try again later.

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

The *pwd* command prints the path name of the working (current) directory.

SEE ALSO

cd(1).

NAME

random − generate a random number

SYNOPSIS

random [−s] [*scale*]

DESCRIPTION

The *random* command generates a random number, *scale*, on the standard output and returns the number as its exit value. By default, this number is either 0 or 1. If *scale* is given a value between 1 and 255, then the range of the random value is from 0 to *scale*. If *scale* is greater than 255, an error message is printed.

When the **−s** (silent) option is given, then the random number is returned as an exit value but is not printed on the standard output. If an error occurs, *random* returns an exit value of zero.

SEE ALSO

rand(3C) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NOTES

This command does not perform any floating-point computations.

random uses the time of day as a seed.

NAME

rc0 – run commands performed to stop the operating system

SYNOPSIS

/etc/rc0

DESCRIPTION

This file is executed at each system state change that needs to have the system in an inactive state. It is responsible for those actions that bring the system to a quiescent state, traditionally called "shutdown."

One system state requires this procedure: state 0 (the system halt state). Whenever a change to this state occurs, the */etc/rc0* procedure is run. The entry in **/etc/inittab** might read:

s0:0:wait:/etc/rc0 >/dev/console 2>&1 </dev/console

Some of the actions performed by */etc/rc0* are carried out by files in the directory **/etc/shutdown.d.** and files beginning with **K** in **/etc/rc0.d**. These files are executed in ASCII order (see FILES below for more information), terminating some system service. The combination of commands in */etc/rc0* and files in **/etc/shutdown.d** and **/etc/rc0.d** determines how the system is shut down.

The recommended sequence for */etc/rc0* is:

Stop System Services and Daemons.

> Various system services (such as Remote File Sharing or LP Spooler) are gracefully terminated.
>
> When new services are added that should be terminated when the system is shut down, the appropriate files are installed in **/etc/shutdown.d** and **/etc/rc0.d.**

Terminate Processes

> SIGTERM signals are sent to all running processes by *killall*(1M). Processes stop themselves cleanly if sent SIGTERM.

Kill Processes

> SIGKILL signals are sent to all remaining processes; no process can resist SIGKILL.
>
> At this point the only processes left are those associated with */etc/rc0* and processes 0 and 1, which are special to the operating system.

Unmount All File Systems

> Only the root file system (/) remains mounted.

FILES

The execution by **/bin/sh** of any files in **/etc/shutdown.d** occurs in ASCII sort-sequence order. See *rc2*(1M) for more information.

SEE ALSO

killall(1M), rc2(1M), shutdown(1M).

NAME

rc2 – run commands performed for multiuser environment

SYNOPSIS

/etc/rc2

DESCRIPTION

This file is executed via an entry in **/etc/inittab** and is responsible for those initializations that bring the system to a ready-to-use state, traditionally state 2, called the "multiuser" state.

The actions performed by */etc/rc2* are found in files in the directory **/etc/rc.d** and files beginning with **S** in **/etc/rc2.d**. These files are executed by **/bin/sh** in ASCII sort–sequence order (see FILES for more information). When functions are added that need to be initialized when the system goes multiuser, an appropriate file should be added in **/etc/rc2.d**.

The functions done by */etc/rc2* command and associated **/etc/rc2.d** files include:

Setting and exporting the TIMEZONE variable.

Setting up and mounting the user (**/usr**) file system.

Cleaning up (remaking) the **/tmp** and **/usr/tmp** directories.

Loading the network interface and ports cards with program data and starting the associated processes.

Starting the *cron* daemon by executing **/etc/cron**.

Cleaning up (deleting) uucp locks status and temporary files in the **/usr/spool/uucp** directory.

Other functions can be added, as required, to support the addition of hardware and software features.

EXAMPLES

The following are prototypical files found in **/etc/rc2.d**. These files are prefixed by an **S** and a number indicating the execution order of the files.

MOUNTFILESYS

```
#   Set up and mount file systems

cd /
/etc/mountall /etc/fstab
```

RMTMPFILES

```
# clean up /tmp
rm -rf /tmp
mkdir /tmp
chmod 777 /tmp
chgrp sys /tmp
chown sys /tmp
```

uucp

```
#   clean-up uucp locks, status, and temporary files

rm -rf /usr/spool/locks/*
```

The file **/etc/TIMEZONE** is included early in */etc/rc2*, thus establishing the default time zone for all commands that follow.

FILES

Here are some hints about files in **/etc/rc.d**:

The order in which files are executed is important. Since they are executed in ASCII sort-sequence order, using the first character of the file name as a sequence indicator will help keep the proper order. Thus, files starting with the following characters would be:

[0-9].	very early
[A-Z].	early
[a-n].	later
[o-z].	last

3.mountfs

Files in **/etc/rc.d** that begin with a dot (.) will not be executed. This feature can be used to hide files that are not to be executed for the time being without removing them. The command can be used only by the super-user.

Files in **/etc/rc2.d** must begin with an **S** or a **K** followed by a number and the rest of the file name. Upon entering run level 2, files beginning with **S** are executed with the **start** option; files beginning with **K** are executed with the **stop** option. Files beginning with other characters are ignored.

SEE ALSO

shutdown(1M).

NAME

relogin – rename login entry to show current layer

SYNOPSIS

/usr/lib/layersys/relogin [–s] [line]

DESCRIPTION

The *relogin* command changes the terminal *line* field of a user's *utmp*(4) entry to the name of the windowing terminal layer attached to standard input. *write*(1) messages sent to this user are directed to this layer. In addition, the *who*(1) command will show the user associated with this layer. The *relogin* command may only be invoked under *layers*(1).

relogin is invoked automatically by *layers*(1) to set the *utmp*(4) entry to the terminal line of the first layer created upon startup and to reset the *utmp*(4) entry to the real line on termination. It may be invoked by a user to designate a different layer to receive *write*(1) messages.

–s Suppress error messages.

line Specifies which *utmp*(4) entry to change. The *utmp*(4) file is searched for an entry with the specified *line* field. That field is changed to the line associated with the standard input. To learn what lines are associated with a given user, say **jdoe**, type:

ps –f –u jdoe

and note the values shown in the **TTY** field [see *ps*(1)].

FILES

/etc/utmp data base of users versus terminals

EXIT STATUS

Returns **0** upon successful completion, **1** otherwise.

SEE ALSO

layers(1), mesg(1), ps(1), who(1), write(1).

utmp(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NOTES

If *line* does not belong to the user issuing the *relogin* command or standard input is not associated with a terminal, *relogin* will fail.

NAME

removepkg — remove installed package

SYNOPSIS

removepkg [software_package]

DESCRIPTION

The *removepkg* command will remove the software package specified as an argument to *removepkg* or will remove the software package the user selects if no argument is given to *removepkg*.

If an argument is specified, *removepkg* will search the list of previously installed packages and remove the first name it matchs. If no name is matched, the user is given an error message.

If no argument is specified, *removepkg* will query the user, via a menu, which package to remove.

You will need to be **root** to remove some packages.

LIMITATIONS

You must invoke *removepkg* on the console.

SEE ALSO

displaypkg(1), installpkg(1).

NAME

restore – restore file to original directory

SYNOPSIS

restore [–c] [–i] [–o] [–t] [–d <device> **]**
[pattern [pattern]...]

DESCRIPTION

–c complete restore. All files on the tape are restored.

–i gets the index file off of the medium. This only works when the archive was created using *backup*. The output is a list of all the files on the medium. No files are actually restored.

–o overwrite existing files. If the file being restored already exists it will not be restored unless this option is specified.

–t indicates that the tape device is to be used. MUST be used with the **–d** option when restoring from tape.

–d *<device>* is the raw device to be used. It defaults to **/dev/rdsk/f0q15d** (the 1.2M floppy).

When doing a restore, one or more patterns can be specified. These patterns are matched against the files on the tape. When a match is found, the file is restored. Since backups are done using full path-names, the file is restored to its original directory. Metacharacters can be used to match multiple files. The patterns should be in quotes to prevent the characters from being expanded before they are passed to the command. If no patterns are specified, it defaults to restoring all files. If a pattern does not match any file on the tape, a message is printed.

When end of medium is reached, the user is prompted for the next media. The user can exit at this point by typing "q". (This may cause files to be corrupted if a file happens to span a medium.) In general, quitting in the middle is not a good idea.

If the file already exists and an attempt is made to restore it without the **–o** option, the file name will be printed on the screen followed by a question mark. This file will not be restored.

In order for multi-volume restores to work correctly, the raw device MUST be used.

SEE ALSO

qt(7), sh(1).

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [–f] [–i] file ...

rm –r [–f] [–i] dirname ... [file ...]

rmdir [–p] [–s] dirname ...

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. If a directory is writable and has the sticky bit set, files within that directory can be removed only if one or more of the following is true [see *unlink*(2)]:

the user owns the file
the user owns the directory
the file is writable by the user
the user is the super-user

If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with **y** (for yes), the file is deleted, otherwise the file remains.

Note that if the standard input is not a terminal, the command will operate as if the **–f** option is in effect.

The *rmdir* command removes the named directory if it is empty and the parent directory is writable. If the parent directory has the sticky bit set, removal occurs only if one of the following is true:

the parent directory is owned by the user
the *dirname* directory is owned by the user
the *dirname* directory is writable to the user
the user is the super-user

Three options apply to *rm*:

–f This option causes the removal of all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory was attempted, this option cannot suppress an error message.

–r This option causes the recursive removal of any directories and subdirectories in the argument list. The directory will be emptied of files and removed. Note that the user is normally prompted for removal of any write-protected files which the directory contains. The write-protected files are removed without prompting, however, if the **–f** option is used, or if the standard input is not a terminal and the **–i** option is not used.

If the removal of a non-empty, write-protected directory was attempted, the command will always fail (even if the **–f** option is used), resulting in an error message.

−i With this option, confirmation of removal of any write-protected file occurs interactively. It overrides the **−f** option and remains in effect even if the standard input is not a terminal.

Two options apply to *rmdir*:

−p This option allows users to remove the directory *dirname* and its parent directories which become empty. A message is printed on standard output as to whether the whole path is removed or part of the path remains for some reason.

−s This option is used to suppress the message printed on standard error when **−p** is in effect.

DIAGNOSTICS

All messages are generally self-explanatory.

It is forbidden to remove the files "." and ".." in order to avoid the consequences of inadvertently doing something like the following:

rm −r .*

Both *rm* and *rmdir* return exit codes of 0 if all the specified directories are removed successfully. Otherwise, they return a non-zero exit code.

SEE ALSO

chmod(1).

rmdir(2), unlink(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

rmail – handle remote mail received via *uucp*

SYNOPSIS

rmail user ...

DESCRIPTION

The *rmail* program interprets incoming mail received via *uucp*(1C). It collapses "From" lines in the form generated by *mail*(1) into a single line of the form "return-path!sender" and passes the processed mail on to *sendmail*(8).

The *rmail* program is explicitly designed for use with *uucp* and *sendmail*.

SEE ALSO

mail(1), uucp(1), sendmail(8).

NOTES

The *rmail* program must be listed in the *uucp* Permissions file for incoming *uucp* mail to work.

NAME

runacct – run daily accounting

SYNOPSIS

/usr/lib/acct/runacct [mmdd [state]]

DESCRIPTION

runacct is the main daily accounting shell procedure. It is normally initiated via *cron*(1M). *runacct* processes connect, fee, disk, and process accounting files. It also prepares summary files for *prdaily* or billing purposes. *runacct* is distributed only to source code licensees.

runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into **active**. When an error is detected, a message is written to **/dev/console**, mail [see *mail*(1)] is sent to **root** and **adm**, and *runacct* terminates. *runacct* uses a series of lock files to protect against re-invocation. The files **lock** and **lock1** are used to prevent simultaneous invocation, and **lastdate** is used to prevent more than one invocation per day.

runacct breaks its processing into separate, restartable *states* using **statefile** to remember the last *state* completed. It accomplishes this by writing the *state* name into **statefile**. *runacct* then looks in **statefile** to see what it has done and to determine what to process next. *States* are executed in the following order:

SETUP	Move active accounting files into working files.
WTMPFIX	Verify integrity of **wtmp** file, correcting date changes if necessary.
CONNECT1	Produce connect session records in **ctmp.h** format.
CONNECT2	Convert **ctmp.h** records into **tacct.h** format.
PROCESS	Convert process accounting records into **tacct.h** format.
MERGE	Merge the connect and process accounting records.
FEES	Convert output of *chargefee* into **tacct.h** format and merge with connect and process accounting records.
DISK	Merge disk accounting records with connect, process, and fee accounting records.
MERGETACCT	Merge the daily total accounting records in **daytacct** with the summary total accounting records in **/usr/adm/acct/sum/tacct**.
CMS	Produce command summaries.
USEREXIT	Any installation-dependent accounting programs can be included here.
CLEANUP	Cleanup temporary files and exit.

To restart *runacct* after a failure, first check the **active** file for diagnostics, then fix up any corrupted data files such as **pacct** or **wtmp**. The **lock** files and **lastdate** file must be removed before *runacct* can be restarted. The argument *mmdd* is necessary if *runacct* is being restarted, and specifies the month and day for which *runacct* will rerun the accounting. Entry point for processing is based on the contents of **statefile**; to override this, include the desired *state* on the command line to designate where processing should begin.

EXAMPLES

To start *runacct.*

```
nohup runacct 2> /usr/adm/acct/nite/fd2log &
```

To restart *runacct.*

```
nohup runacct 0601 2>> /usr/adm/acct/nite/fd2log &
```

To restart *runacct* at a specific *state.*

```
nohup runacct 0601 MERGE 2>> /usr/adm/acct/nite/fd2log &
```

FILES

/etc/wtmp
/usr/adm/pacct*
/usr/src/cmd/acct/tacct.h
/usr/src/cmd/acct/ctmp.h
/usr/adm/acct/nite/active
/usr/adm/acct/nite/daytacct
/usr/adm/acct/nite/lock
/usr/adm/acct/nite/lock1
/usr/adm/acct/nite/lastdate
/usr/adm/acct/nite/statefile
/usr/adm/acct/nite/ptacct*.*mmdd*

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), mail(1).
acct(2), acct(4), utmp(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

BUGS

Normally, it is not a good idea to restart *runacct* in the **SETUP** *state*. Run **SETUP** manually and restart via:

runacct *mmdd* **WTMPFIX**

If *runacct* failed in the **PROCESS** *state,* remove the last **ptacct** file because it will not be complete.

NAME

sag − system activity graph

SYNOPSIS

sag [options]

DESCRIPTION

The *sag* command graphically displays the system activity data stored in a binary data file by a previous *sar*(1) run. Any of the *sar* data items may be plotted singly, or in combination; as cross plots, or versus time. Simple arithmetic combinations of data may be specified. The *sag* command invokes *sar* and finds the desired data by string-matching the data column header (run *sar* to see what is available). These *options* are passed through to *sar*:

−s *time* Select data later than *time* in the form *hh* [*:mm*]. Default is 08:00.

−e *time* Select data up to *time*. Default is 18:00.

−i *sec* Select data at intervals as close as possible to *sec* seconds.

−f *file* Use *file* as the data source for *sar*. Default is the current daily data file **/usr/adm/sa/sa***dd*.

Other *options*:

−T *term* Produce output suitable for terminal *term*. See *tplot*(1G) for known terminals. Default for *term* is **$TERM**.

−x *spec* x axis specification with *spec* in the form:
"name [op name]... [lo hi]"

−y *spec* y axis specification with *spec* in the same form as above.

Name is either a string that will match a column header in the *sar* report, with an optional device name in square brackets, e.g., **r+w/s[dsk−1]**, or an integer value. *Op* is + − * or / surrounded by blanks. Up to five names may be specified. Parentheses are not recognized. Contrary to custom, + and − have precedence over * and /. Evaluation is left to right. Thus A / A + B * 100 is evaluated (A/(A+B))*100, and A + B / C + D is (A+B)/(C+D). *Lo* and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

A single *spec* is permitted for the x axis. If unspecified, *time* is used. Up to 5 *spec*'s separated by ; may be given for **−y**. Enclose the **−x** and **−y** arguments in "" if blanks or \<CR> are included. The **−y** default is:

−y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"

EXAMPLES

To see today's CPU utilization:

```
sag
```

To see activity over 15 minutes of all disk drives:

```
TS=date +%H:%M
sar −o tempfile 60 15
TE=date +%H:%M
sag −f tempfile −s $TS −e $TE −y "r+w/s[dsk]"
```

FILES

/usr/adm/sa/sa*dd* daily data file for day *dd*

SEE ALSO

sar(1), tplot(1G).

NAME

sar – system activity reporter

SYNOPSIS

sar [–ubdycwaqvmprDSAC] [–o file**]** t **[** n **]**

sar [–ubdycwaqvmprDSAC] [–s time**] [–e** time**] [–i** sec**]**
[–f file**]**

DESCRIPTION

sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds, where *t* should be 5 or greater. If the **–o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, *sar* extracts data from a previously recorded *file*, either the one specified by the **–f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd*. The starting and ending times of the report can be bounded via the **–s** and **–e** *time* arguments of the form *hh*[:*mm*[:*ss*]]. The **–i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

–u Report CPU utilization (the default):
%usr, %sys, %wio, %idle – portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle. When used with **–D**, %sys is split into percent of time servicing requests from remote machines (%sys remote) and all other system time (%sys local).

–b Report buffer activity:
bread/s, bwrit/s – transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s – accesses of system buffers;
%rcache, %wcache – cache hit ratios, i. e., (1–bread/lread) as a percentage;
pread/s, pwrit/s – transfers via raw (physical) device mechanism. When used with **–D**, buffer caching is reported for locally-mounted remote resources.

–d Report activity for each block device, e. g., disk or tape drive. When data is displayed, the device specification *dsk-* is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent. The activity data reported is:
%busy, avque – portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s – number of data transfers from or to device, number of bytes transferred in 512-byte units;
avwait, avserv – average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency, and data transfer times).

–y Report TTY device activity:
rawch/s, canch/s, outch/s – input character rate, input character rate processed by canon, output character rate;

rcvin/s, xmtin/s, mdmin/s – receive, transmit and modem interrupt rates.

–c Report system calls:
scall/s – system calls of all types;
sread/s, swrit/s, fork/s, exec/s – specific system calls;
rchar/s, wchar/s – characters transferred by read and write system calls. When used with **–D**, the system calls are split into incoming, outgoing, and strictly local calls.

–w Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s – number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs);
pswch/s – process switches.

–a Report use of file access system routines:
iget/s, namei/s, dirblk/s.

–q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc – run queue of processes in memory and runnable;
swpq-sz, %swpocc – swap queue of processes swapped out but ready to run.

–v Report status of process, inode, file tables:
text-sz, proc-sz, inod-sz, file-sz, lock-sz – entries/size for each table, evaluated once at sampling point;
ov – overflows that occur between sampling points for each table.

–m Report message and semaphore activities:
msg/s, sema/s – primitives per second.

–p Report paging activities:
vflt/s – address translation page faults (valid page not in memory);
pflt/s – page faults from protection errors (illegal access to page) or "copy-on-writes";
pgfil/s – vflt/s satisfied by page-in from file system;
rclm/s – valid pages reclaimed for free list.

–r Report unused memory pages and disk blocks:
freemem – average pages available to user processes;
freeswap – disk blocks available for process swapping.

–D Report Remote File Sharing activity:
When used in combination with **–u**, **–b**, or **–c**, it causes *sar* to produce the remote file sharing version of the corresponding report. **–Du** is assumed when only **–D** is specified.

–S Report server and request queue status:
Average number of Remote File Sharing servers on the system (serv/lo-hi), % of time receive descriptors are on the request queue (request %busy), average number of receive descriptors waiting for service when queue is occupied (request avg lgth), % of time there are idle servers (server %avail), average number of idle servers when idle ones exist (server avg avail).

−A Report all data. Equivalent to **−udqbwcayvmprSDC**.

−C Report Remote File Sharing buffer caching overhead:
snd-inv/s - number of invalidation messages per second sent by your machine as a server.
snd-msg/s - total outgoing RFS messages sent per second.
rcv-inv/s - number of invalidation messages received from the remote server.
rcv-msg/s - total number of incoming RFS messages received per second.
dis-bread/s - number of buffer reads that would be eligible for caching if caching were not turned off. (Indicates the penalty of running uncached.)
blk-inv/s - number of buffers removed from the client cache.

EXAMPLES

To see today's CPU activity so far:

    sar

To watch CPU activity evolve for 10 minutes and save data:

    sar −o temp 60 10

To later review disk and tape activity from that period:

    sar −d −f temp

FILES

/usr/adm/sa/sa*dd* daily data file, where *dd* are digits representing the day of the month

SEE ALSO

sag(1G), sar(1M).

NAME

sar: 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 [**−ubdycwaqvmprDSAC**] [**−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 several counters that are incremented as various system actions occur. These include counters for CPU utilization, buffer usage, disk and tape I/O activity, TTY device activity, switching and system-call activity, file-access, queue activity, inter-process communications, paging, and Remote File Sharing.

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, with an interval of *t* seconds between samples, and writes in binary format to *ofile* or to standard output. The sampling interval *t* should be greater than 5 seconds; otherwise, the activity of *sadc* itself may affect the sample. If *t* and *n* are omitted, a special record is written. This facility is used at system boot time, when booting to a multiuser state, to mark the time at which the counters restart from zero. For example, the **/etc/init.d/perf** file writes the restart mark to the daily data by the command entry:

```
su sys −c "/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`"
```

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file **/usr/adm/sa/sa***dd* 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 **/usr/spool/cron/crontabs/sys** [see *cron*(1M)]:

```
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8−17 * * 1−5 /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***dd*. The options are explained in *sar*(1). The **/usr/spool/cron/crontabs/sys** entry:

```
5 18 * * 1−5 /usr/lib/sa/sa2 −s 8:00 −e 18:01 −i 1200 −A
```

will report important activities hourly during the working day.

The structure of the binary daily data file is:

```
struct sa {
        struct sysinfo si;  /* see /usr/include/sys/sysinfo.h */
        struct minfo mi;  /* defined in sys/sysinfo.h */
        struct dinfo di;    /* RFS info defined in sys/sysinfo.h */
        struct rcinfo rc;  /* Client cache info defined in sys/sysinfo.h */
        struct bpbinfo bi; /* Co-processor info defined in sys/sysinfo.h */
        int bpb_utilize   /* Co-processor utilize flag */
        int minserve, maxserve;    /* RFS server low and high water marks */
        int  szinode;       /* current size of inode table  */
        int  szfile;        /* current size of file table  */
        int  szproc;        /* current size 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 inode table  */
        int  mszfile;       /* size of file 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;     /* cumulative overflows of inode table  */
        long  fileovf;      /* cumulative overflows of file table  */
        long  procovf;      /* cumulative overflows of proc table  */
        time_t  ts;         /* time stamp, seconds  */
        long  devio[NDEVS][4];     /* device unit information  */
#define IO_OPS          0          /* cumulative I/O requests  */
#define IO_BCNT         1          /* cumulative blocks transferred */
#define IO_ACT          2          /* cumulative drive busy time in ticks  */
#define IO_RESP         3          /* cumulative I/O resp time in ticks */
};
```

FILES

/usr/adm/sa/sa*dd*	daily*data*file
/usr/adm/sa/sar*dd*	daily*report*file
/tmp/sa.adrfl	**addressfile**

SEE ALSO

cron(1M), sag(1G), sar(1), timex(1).

NAME

sdiff – side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

The *sdiff* command 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).

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\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

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)**l** List the pattern space on the standard output in an unambiguous form. Non-printable characters are displayed in octal notation 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.

(1)**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 *regular expression.*
- **g** Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
- **p** Print the pattern space if a replacement was made.

w *wfile* Write. Append the pattern space to *wfile* 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.

(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

setcolor – set screen color

SYNOPSIS

setcolor [–brgfopcs] argument [,argument]

DESCRIPTION

The *setcolor* utility allows the user to set various parameters on a color display, including foreground and background colors, overscan and reverse video colors, bell characteristics, and hardware scrolling. A range of 16 colors can be used for the foreground color; a range of 8 colors can be used for the background color. Invoking *setcolor* with no arguments produces a usage message and displays the 16 available colors.

The available colors that can be specified to *setcolor* are:

blue	magenta	brown	black
lblue	lmagenta	yellow	grey
cyan	white	green	red
lcyan	lwhite	lgreen	lred

Arguments:

–b *color* Set the background to the specified color where *color* is a string taken from the above list.

–r *color* [, *color*]
Set the reverse video characters' foreground color to the first color. If a second choice is specified, set the reverse video characters' background to the specified color.

–g *color* [,*color*]
Set the graphics characters' foreground color to the first color. If a second choice is specified, set the graphics characters' background to the second color.

–f *color* [,*color*]
Set the foreground to the first color. If a second choice is specified, set the background to the second color.

–o *color* Set the overscan (screen border) color to the specified color.

–p *frequency duration*
Set the frequency and duration of the bell. The duration is specified in $1/10^{th}$-second increments. The frequency is the period in microseconds.

–c *top bottom*
Set the top and bottom scan lines of the cursor. Depending on the font being displayed, the size of the cursor could be 8, 14, or 16 scan lines.

–s *on/off* Set the hardware scrolling feature to be on or off. The *display*(7) driver provides for hardware-assisted scrolling on CGA, EGA, and VGA displays for better output performance. The standard system is shipped with hardware scrolling turned *off*. This option to *setcolor* allows the user to turn hardware scrolling on when desired.

Notes

The *setcolor* utility functions only on display devices capable of supporting the ANSI escape sequences for color codes. It generally displays the message:

setcolor: unsupported terminal type

when run on unsupported hardware.

SEE ALSO

display(7).

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

setmnt – establish mount table

SYNOPSIS

/etc/setmnt

DESCRIPTION

The *setmnt* command creates the **/etc/mnttab** table, which is needed for both the *mount*(1M) and *umount* commands. The *setmnt* command 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., **/dev/dsk/1s1**) and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the mount table entry.

FILES

/etc/mnttab

SEE ALSO

mount(1M).

BUGS

Problems may occur if *filesys* or *node* are longer than 32 characters. The *setmnt* command silently enforces an upper limit on the maximum number of *mnttab* entries.

NAME

settime – change a files access and modification dates

SYNOPSIS

settime *mmddhhmm* **[** *yy* **] [–f** *fname* **]** *name* ...

DESCRIPTION

settime sets the access and modification dates for one or more files. The dates are set to the specified date or to the access and modification dates of the file specified via **–f**. Exactly one of these methods must be used to specify the new date(s). 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 two digits of the year and is optional. For example:

settime 1008004583 ralph pete

sets the access and modification dates of files *ralph* and *pete* to Oct 8, 12:45 AM, 1983. Another example:

settime –f ralph john

This sets the access and modification dates of the file *john* to those of the file *ralph*.

SEE ALSO

date(1), touch(1).

NOTES

Use of *touch* in place of *settime* is encouraged.

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 **|**. 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 newlines 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 executed in the current (that is, parent) shell.

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* below).

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 shell reads commands from the string between two grave accents (``) and the standard output from these commands may be used as all or part of a word. Trailing new-lines from the standard output are removed.

No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (`) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes (" ...` ...` ... "), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a new-line character (**\new-line**), both the backslash and the new-line are removed (see the later section on *Quoting*). In addition, backslashes used to escape dollar signs (**\$**) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \, `, ", **new-line**, and **$** are left intact when the command string is read.

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 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.

SHELL When the shell is invoked, it scans the environment (see *Environment* below) for this name. If it is found and 'rsh' is 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.

Input/Output

A command's 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 as arguments to the invoked command. Note that parameter and 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* After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file. If, however, − is appended to <<:

(1) leading tabs are stripped from *word* before the shell input is read (but after parameter and command substitution is done on *word*),

(2) leading tabs are stripped from the shell input as it is read and before each line is compared with *word*, and

(3) shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file.

If any character of *word* is quoted (see *Quoting* below), no additional processing is done to the shell input. If no characters of *word* are quoted:

(1) parameter and command substitution occurs,

(2) (escaped) **\new-line** is ignored, and

(3) \ must be used to quote the characters \, $, and `.

The resulting document becomes the standard input.

<&*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*.

Using the terminology introduced on the first page, under *Commands*, if a *command* is composed of several *simple-commands*, redirection will be evaluated for the entire *command* before it is evaluated for each *simple-command*. That is, the shell evaluates redirection for the entire *list*, then each *pipeline* within the *list*, then each *command* within each *pipeline*, then each *list* within each *command*.

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.

File Name Generation

Before a command is executed, 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 backslash (\) or inserting it between a pair of quote marks (´´ or ""). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair **\new-line** is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (´´), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, "´").

Inside a pair of double quote marks (""), parameter and command substitution occurs and the shell quotes the results to avoid blank interpretation and file name generation. If **$*** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces (**"$1 $2** ..."); however, if **$@** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces (**"$1" "$2"** ...). \ quotes the characters \, `, ", and **$**. The pair **\new-line** is removed before parameter and command substitution. If a backslash precedes characters other than \, `, ", **$**, and new-line, then the backslash itself is quoted by the shell.

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.

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 *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd                                  and
(export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **−k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name.

The following first prints **a=b c** and **c**:

```
echo a=b c
set −k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign, between two colon delimiters anywhere in the path list, or at the end of the path list. If the command name contains a /, the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash −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 [**−n**] [*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, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are *not* exported.

getopts Use in shell scripts to support command syntax standards [see *intro*(1)]; it parses positional parameters and checks for legal options. See *getopts*(1) for usage and description.

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. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is 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*(1M) for usage and description.

pwd
: Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]
: One line is read from the standard input and, using the internal field separator, **IFS** (normally space or tab), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. Lines can be continued using **\new-line**. Characters other than **new-line** can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
: 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 [**−−aefhkntuvx** [*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 [*n*]
: Impose a size limit of *n* blocks on files written by the shell and its child processes (files of any size may be read). If *n* is omitted, the current limit is printed. You may lower your own ulimit, but only a super-user [see *su*(1M)] can raise a ulimit.

umask [*nnn*]
: The user file-creation mask is set to *nnn* [see *umask*(1)]. 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 your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be 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 TERMINATE 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.

A restricted shell can be invoked in one of the following ways: (1) *rsh* is the file name part of the last entry in the **/etc/passwd** file [see *passwd*(4)]; (2) the environment variable **SHELL** exists and *rsh* is the file name part of its value; (3) the shell is invoked and *rsh* is the file name part of argument 0; (4) the shell is invoked with the **−r** option.

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** [see *profile*(4)] 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 a restricted shell. 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

cd(1), echo(1), env(1), getopts(1), intro(1), login(1), newgrp(1M), pwd(1), test(1), umask(1), wait(1).
dup(2), exec(2), fork(2), pipe(2), profile(4), signal(2), ulimit(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

CAVEATS

Words used for file names in input/output redirection are not interpreted for file name generation (see *File Name Generation* above). For example, **cat file1 >a*** will create a file named **a***.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message *cannot fork, too many processes*, try using the *wait*(1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

BUGS

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.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For **wait** *n* , if *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

shl – shell layer manager

SYNOPSIS

shl

DESCRIPTION

The *shl* command 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 **swtch** (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), sxt(7).
ioctl(2), signal(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

shutdown — shut down system, change system state

SYNOPSIS

/etc/shutdown [**−y**] [**−g**grace_period [**−i**init_state]

DESCRIPTION

This command is executed by the super-user 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 command can be executed from the console only. This 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 confirmation question so the command can be run without user intervention. A default of 60 seconds is allowed between the warning message and the final message. Another 60 seconds is allowed between the final message and the confirmation.

−g*grace_period*
allows the super-user to change the number of seconds from the 60-second default.

−i*init_state*
specifies the state that *init*(1M) is to be put in following the warnings, if any. By default, system state "**s**" is used (the same as states "**1**" and "**S**").

Other recommended system state definitions are:

state 0
Shut the machine down so it is safe to remove the power. Have the machine remove power if it can. The */etc/rc0* procedure is called to do this work.

state 1, s, S
Bring the machine to the state traditionally called single-user. The */etc/rc0* procedure is called to do this work. (Though **s** and **1** are both used to go to single user state, **s** only kills processes spawned by *init* and does not unmount file systems. State **1** unmounts everything except root and kills all user processes, except those that relate to the console.)

state 6
Stop the UNIX system and reboot to the state defined by the *initdefault* entry in **/etc/inittab**.

SEE ALSO

init(1M), rc0(1M), rc2(1M).
inittab(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

The *sleep* command 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 *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

sort – sort and/or merge files

SYNOPSIS

sort [–cmu] [–ooutput] [–ykmem] [–zrecsz] [–dfiMnr] [–btx]
[+pos1 [–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*.

–y*kmem*
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, **–y**0 is guaranteed to start with minimum memory. By convention, **–y** (with no argument) starts with maximum memory.

–z*recsz*
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 non-printable characters.

−M Compare as months. The first three non-blank characters of the field are folded to upper case and compared. For example, in English the sorting order is "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The **−M** option implies the **−b** option (see the following text).

−n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **−n** option implies the **−b** option (see the following text). 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 in the following text), 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 just before *pos2*. The characters at position *pos1* and just before *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:

−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).

−t*x* 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 (for example, *xx* delimits an empty field).

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfinr**. 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*.0**b** 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*.1**b** 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).

WARNINGS

Comments and exits with non-zero status for various trouble conditions (for example, 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.

sort does not guarantee preservation of relative line ordering on equal keys.

NAME

spell, hashmake, spellin, hashcheck − find spelling errors

SYNOPSIS

spell [**−v**] [**−b**] [**−x**] [**−l**] [+local_file] [files]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

The *spell* command 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.

The *spell* command 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*.

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 *+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+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell*:

hashmake Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.

spellin Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.

hashcheck Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/usr/lib/spell/spellhist	history file
/usr/lib/spell/spellprog	program

SEE ALSO

deroff(1), sed(1), sort(1), tee(1).

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*.

NAME

spline – interpolate smooth curve

SYNOPSIS

spline [options]

DESCRIPTION

The *spline* command takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic *spline* output has two continuous derivatives and sufficient points to look smooth when plotted, for example, by *graph*(1G).

The following *options* are recognized, each as a separate argument:

–a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

–k The constant k used in the boundary value computation:

$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$

is set by the next argument (default $k = 0$).

–n Space output points so that approximately n intervals occur between the lower and upper x limits (default $n = 100$).

–p Make output periodic, i.e., match derivatives at ends. First and last input values should normally agree.

–x Next 1 (or 2) arguments are lower (and upper) x limits. Normally, these limits are calculated from the data. Automatic abscissas start at lower limit (default 0).

SEE ALSO

graph(1G).

DIAGNOSTICS

When data is not strictly monotone in x, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1,000 input points is enforced silently.

NAME

split − split a file into pieces

SYNOPSIS

split [−n] [file [name]]

DESCRIPTION

The *split* command 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 instead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

NAME

strace – print STREAMS trace messages

SYNOPSIS

strace [mid sid level] ...

DESCRIPTION

The *strace* command without arguments writes all STREAMS event trace messages from all drivers and modules to its standard output. These messages are obtained from the STREAMS log driver [*log*(7)]. If arguments are provided they must be in triplets of the form *mid, sid, level*, where *mid* is a STREAMS module id number, *sid* is a sub-id number, and *level* is a tracing priority level. Each triplet indicates that tracing messages are to be received from the given module/driver, sub-id (usually indicating minor device), and priority level equal to or less than the given level. The token **all** may be used for any member to indicate no restriction for that attribute.

The format of each trace message output is:

<seq> <time> <ticks> <level> <flags> <mid> <sid> <text>

<seq>	trace sequence number
<time>	time of message in hh:mm:ss
<ticks>	time of message in machine ticks since boot
<level>	tracing priority level
<flags>	**E** : message is also in the error log **F** : indicates a fatal error **N** : mail was sent to the system administrator
<mid>	module id number of source
<sid>	sub-id number of source
<text>	formatted text of the trace message

Once initiated, *strace* will continue to execute until terminated by the user.

EXAMPLES

Output all trace messages from the module or driver whose module id is 41:

    strace 41 all all

Output those trace messages from driver/module id 41 with sub-ids 0, 1, or 2:

    strace 41 0 1 41 1 1 41 2 0

Messages from sub-ids 0 and 1 must have a tracing level less than or equal to 1. Those from sub-id 2 must have a tracing level of 0.

CAVEATS

Due to performance considerations, only one *strace* process is permitted to open the STREAMS log driver at a time. The log driver has a list of the triplets specified in the command invocation, and compares each potential trace message against this list to decide if it should be formatted and sent up to the *strace* process. Hence, long lists of triplets will have a greater impact on overall STREAMS performance.

Running *strace* will have the most impact on the timing of the modules and drivers generating the trace messages that are sent to the *strace* process. If trace messages are generated faster than the *strace* process can handle them, then some of the messages will be lost. This last case can be determined by examining the sequence numbers on the trace messages output.

SEE ALSO

log(7).
STREAMS Programmer's Guide.

NAME

strclean – STREAMS error logger cleanup program

SYNOPSIS

strclean [**−d** logdir] [**−a** age]

DESCRIPTION

The *strclean* command is used to clean up the STREAMS error logger directory on a regular basis [for example, by using *cron*(1M)]. By default, all files with names matching **error.*** in **/usr/adm/streams** that have not been modified in the last 3 days are removed. A directory other than **/usr/adm/streams** can be specified using the **−d** option. The maximum age in days for a log file can be changed using the **−a** option.

EXAMPLE

```
strclean −d /usr/adm/streams −a 3
```

has the same result as running strclean with no arguments.

NOTES

The *strclean* command is typically run from *cron*(1M) on a daily or weekly basis.

FILES

/usr/adm/streams/error.*

SEE ALSO

cron(1M), strerr(1M).
STREAMS Programmer's Guide.

NAME

strerr – STREAMS error logger daemon

SYNOPSIS

strerr

DESCRIPTION

The *strerr* routine receives error log messages from the STREAMS log driver [*log*(7)] and appends them to a log file. The error log files produced reside in the directory **/usr/adm/streams**, and are named **error.***mm-dd*, where *mm* is the month and *dd* is the day of the messages contained in each log file.

The format of an error log message is:

<seq> <time> <ticks> <flags> <mid> <sid> <text>

<*seq*>	error sequence number
<*time*>	time of message in hh:mm:ss
<*ticks*>	time of message in machine ticks since boot priority level
<*flags*>	**T** : the message was also sent to a tracing process **F** : indicates a fatal error **N** : send mail to the system administrator
<*mid*>	module id number of source
<*sid*>	sub-id number of source
<*text*>	formatted text of the error message

Messages that appear in the error log are intended to report exceptional conditions that require the attention of the system administrator. Those messages which indicate the total failure of a STREAMS driver or module should have the F flag set. Those messages requiring the immediate attention of the administrator will have the N flag set, which causes the error logger to send the message to the system administrator via *mail*(1). The priority level usually has no meaning in the error log but will have meaning if the message is also sent to a tracer process.

Once initiated, *strerr* will continue to execute until terminated by the user. Commonly, *strerr* would be executed asynchronously.

CAVEATS

Only one *strerr* process at a time is permitted to open the STREAMS log driver.

If a module or driver is generating a large number of error messages, running the error logger will cause a degradation in STREAMS performance. If a large burst of messages are generated in a short time, the log driver may not be able to deliver some of the messages. This situation is indicated by gaps in the sequence numbering of the messages in the log files.

FILES

/usr/adm/streams/error.*mm-dd*

SEE ALSO

log(7).
STREAMS Programmer's Guide.

NAME

strings − find the printable strings in an object file

SYNOPSIS

strings [−] [−a] [−o] [*−number* **]** *file* ...

DESCRIPTION

strings looks for ASCII strings in a binary file. A string is any sequence of four or more printing characters ending with a newline or a null character. Unless the − flag is given, *strings* only looks in the initialized data space of object files. If the **−o** flag is given, then each string is preceded by its decimal offset in the file. **-a** works the same as the - flag. If the *−number* flag is given, then *number* is used as the minimum string length rather than 4.

strings is useful for identifying random object files.

SEE ALSO

hd(1).

CREDIT

This utility was developed at the University of California at Berkeley and is used with permission.

NAME

stty — set the options for a terminal

SYNOPSIS

stty [−a] [−g] [options]

DESCRIPTION

The *stty* command sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

In this report, if a character is preceded by a caret (^), then the value of that option is the corresponding CTRL character (e.g., "**^h**" is **CTRL-h** ; in this case, recall that **CTRL-h** is the same as the "back-space" key.) The sequence "**^′**" means that an option has a null value. For example, normally **stty −a** will report that the value of **swtch** is "**^′**"; however, if *shl*(1) or *layers*(1) has been invoked, **stty −a** will have the value "**^z**".

−a Reports all of the option settings.

−g Reports current settings in a form that can be used as an argument to another *stty* command.

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

parenb (−parenb) 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.

110 300 600 1200 1800 2400 4800 9600 19200 38400
Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)

hupcl (−hupcl) Hang up (do not hang up) Dataphone data set connection on last close.

hup (−hup) Same as **hupcl (−hupcl)**.

cstopb (−cstopb) Use two (one) stop bits per character.

cread (−cread) Enable (disable) the receiver.

clocal (−clocal) 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.
iuclc (−iuclc)	Map (do not map) uppercase alphabetics to lowercase 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.
tostop	Background jobs stop if they attempt terminal output.
−tostop	Output from background jobs to the terminal is allowed.

Output Modes

opost (−opost)	Post-process output (do not post-process output; ignore all other output modes).
olcuc (−olcuc)	Map (do not map) lowercase alphabetics to uppercase on output.
onlcr (−onlcr)	Map (do not map) NL to CR-NL on output.
ocrnl (−ocrnl)	Map (do not map) CR to NL on output.
onocr (−onocr)	Do not (do) output CRs at column zero.
onlret (−onlret)	On the terminal NL performs (does not perform) the CR function.
ofill (−ofill)	Use fill characters (use timing) for delays.
ofdel (−ofdel)	Fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	Select style of delay for carriage returns (see *termio*(7)).
nl0 nl1	Select style of delay for line-feeds (see *termio*(7)).
tab0 tab1 tab2 tab3	Select style of delay for horizontal tabs (see *termio*(7)).
bs0 bs1	Select style of delay for backspaces (see *termio*(7)).
ff0 ff1	Select style of delay for form-feeds (see *termio*(7)).
vt0 vt1	Select style of delay for vertical tabs (see *termio*(7)).

Local Modes

isig (−isig)	Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.
icanon (−icanon)	Enable (disable) canonical input (ERASE and KILL processing).
xcase (−xcase)	Canonical (unprocessed) uppercase/lowercase presentation.

echo (**−echo**)	Echo back (do not echo back) every character typed.
echoe (**−echoe**)	Echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (**−echok**)	Echo (do not echo) NL after KILL character.
lfkc (**−lfkc**)	The same as **echok** (**−echok**); obsolete.
echonl (**−echonl**)	Echo (do not echo) NL.
noflsh (**−noflsh**)	Disable (enable) flush after INTR, QUIT, or SWTCH.
stwrap (**−stwrap**)	Disable (enable) truncation of lines longer than 79 characters on a synchronous line.
stflush (**−stflush**)	Enable (disable) flush on a synchronous line after every *write*(2).
stappl (**−stappl**)	Use application mode (use line mode) on a synchronous line.

Control Assignments

control-character c	Set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **swtch**, **eof**, **eol**, **ctab**, **min**, or **time** (**ctab** is used with **−stappl**; **min** and **time** are used with **−icanon**; see *termio*(7)). If *c* 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.
line *i*	Set line discipline to *i* ($0 < i < 127$).

Combination Modes

evenp or **parity**	Enable **parenb** and **cs7**.
oddp	Enable **parenb**, **cs7**, and **parodd**.
−parity, **−evenp**, or **−oddp**	Disable **parenb**, and set **cs8**.
raw (**−raw** or **cooked**)	Enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).
nl (**−nl**)	Unset (set) **icrnl**, **onlcr**. In addition **−nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (**−lcase**)	Set (unset) **xcase**, **iuclc**, and **olcuc**.
LCASE (**−LCASE**)	Same as **lcase** (**−lcase**).
tabs (**−tabs** or **tab3**)	Preserve (expand to spaces) tabs when printing.
ek	Reset ERASE and KILL characters back to normal **#** and **@**.
sane	Resets all modes to some reasonable values.
term	Set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

Control Modes for the Video Monitor

mono Selects the monochrome display as the output device for the console screen. This mode is valid if a standard monochrome adapter is present or if a standard enhanced graphics adapter (EGA) is present and the EGA is currently in one of the monochrome display modes.

color Selects a standard regular color display as the output device for the console screen. This mode is valid if a color graphics adapter is present or if a standard EGA is present and is currently in one of the color graphics compatibility modes.

enhanced Selects the enhanced color display as the output device for the console screen. This mode is valid if an EGA is present and is currently in a non-monochrome display mode.

Control Modes for the Attached Display Devices

B40x25 Selects 40x25 (40 columns x 25 rows) black and white text display mode.

C40x25 Selects 40x25 color text display mode.

B80x25 Selects 80x25 black and white text display mode.

C80x25 Selects 80x25 color display text mode.

BG320 Selects 320x200 black and white graphics display mode.

CG320 Selects 320x200 color graphics display mode.

BG640 Selects 640x200 black and white graphics display mode.

The keyboard and display control modes above are valid for the following configurations: standard color graphics adapter (CGA) attached to an standard regular color display; standard enhanced graphics adapter (EGA) (modes 0-6) attached to a standard regular color display or standard enhanced color display.

CG320_D
Selects EGA support for 320x200 graphics display mode (EGA mode D).

CG640_E
Selects EGA support for 640x200 graphics display mode (EGA mode E).

The two options above are only valid when an EGA is attached to a standard regular color display or an enhanced color display.

EGAMONO80x25
Selects EGA Mode 7 as the display mode. Emulates the support provided by the standard monochrome display adapter.

EGAMONOAPA
Selects EGA support for 640x350 graphics display mode (EGA mode F).

ENHMONOAPA2
Selects EGA mode F*.

The three options above are only valid when a standard EGA is attached to an IBM monochrome display.

ENH_B40x25
Selects enhanced EGA support for 40x25 black and white text display mode (EGA mode 0*).

ENH_C40x25
Selects enhanced EGA support for 40x25 color text display mode (EGA mode 1*).

ENH_B80x25
Selects enhanced EGA support for 80x25 black and white text display mode (EGA mode 2*).

ENH_C80x25
Selects enhanced EGA support for 80x25 color text display mode (EGA mode 3*).

ENH_B80x43
Selects enhanced EGA support for 80x43 black and white text display mode.

ENH_C80x43
Selects enhanced EGA support for 80x43 color text display mode.

CG640x350
Selects EGA support for 640x350 graphics display mode (EGA mode 10).

ENH_CG640
Selects EGA mode 10*.

The options above are only valid when a standard EGA is attached to a standard enhanced color display.

MCAMODE
Reinitializes the monochrome graphics adapter.

ENH_CGA
Selects CGA hardware emulation, when an AT&T Super-Vu video controller is attached.

INTERACTIVE UNIX System Character Mapping Mode

An additional mode is supported on the INTERACTIVE UNIX Operating System:

map *maparg* Run the *ttymap*(1) command with the argument *maparg*. The *ttymap* command sets the terminal character mappings. Multiple argument words can be passed to *ttymap* by enclosing them in quotes.

The following example shows how to set the mapping for a French console:

stty map /usr/lib/keyboard/french.map

To disable the terminal mapping:

stty map −d

SEE ALSO

tabs(1), ttymap(1), termio(7).
ioctl(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

su — become super-user or another user

SYNOPSIS

su [—] [name [arg ...]] —c —r

DESCRIPTION

The *su* command 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/sh**, the user's **.profile** can check *arg0* for **—sh** or **—su** to determine if it was invoked by *login*(1) or *su*(1M), respectively. If the user's program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *-program* by both *login*(1) and *su*(1M).

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:

```
/bin/su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, execute:

```
/bin/su — bin
```

To execute *command* with the temporary environment and permissions of user **bin**, type:

/bin/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 *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

sulogin – access single-user mode

SYNOPSIS

sulogin

DESCRIPTION

sulogin is automatically invoked by *init* when the system is first started. It prompts the user to type the root password to enter system maintenance mode (single-user mode) or to type CTRL-D for normal startup (multi-user mode). *sulogin* should never be directly invoked by the user.

FILES

/etc/sulogin

SEE ALSO

init(M).

NAME

sum – print checksum and block count of a file

SYNOPSIS

sum [**–r**] file

DESCRIPTION

The *sum* command calculates and prints a 16-bit checksum for the named file and 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

swap — swap administrative interface

SYNOPSIS

/etc/swap −a swapdev swaplow swaplen
/etc/swap −d swapdev swaplow
/etc/swap −l

DESCRIPTION

The *swap* command provides a method of adding, deleting, and monitoring the system swap areas used by the memory manager. The following options are recognized:

−a Add the specified swap area. *swapdev* is the name of the block special device, e.g., **/dev/dsk/1s0**. *swaplow* is the offset in 512-byte blocks into the device where the swap area should begin. *swaplen* is the length of the swap area in 512-byte blocks. This option can only be used by the super-user. Swap areas are normally added by the system start-up routine **/etc/rc** when going into multiuser mode.

−d Delete the specified swap area. *swapdev* is the name of a block special device, e.g., **/dev/dsk/1s0**. *swaplow* is the offset in 512-byte blocks into the device where the swap area should begin. This option can only be used by the super-user.

−l List the status of all the swap areas. The output has four columns:

- **DEV** The *swapdev* special file for the swap area if one can be found in the **/dev/dsk** or **/dev** directories, and its major/minor device number in decimal.
- **LOW** The *swaplow* value for the area in 512-byte blocks.
- **LEN** The *swaplen* value for the area in 512-byte blocks.
- **FREE** The number of free 512-byte blocks in the area.

WARNINGS

No check is done to see if a swap area being added overlaps with an existing swap area or file system.

NAME

sync — update the super block

SYNOPSIS

sync

DESCRIPTION

The *sync* command 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.

NOTE

If you have done a write to a file on a remote machine in a Remote File Sharing environment, you cannot use *sync* to force buffers to be written out to disk on the remote machine. *sync* will only write local buffers to local disks.

SEE ALSO

sync(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

sysdef – output values of tunable parameters

SYNOPSIS

/etc/sysdef [system_namelist [conf]]

DESCRIPTION

The *sysdef* command outputs the values of all tunable parameters. It generates the output by analyzing the named operating system file (*system_namelist*) and extracting the configuration information from the name list itself.

FILES

/unix	default operating system file (where the system namelist is)
/etc/conf/*	default directory containing master files

SEE ALSO

nlist(3C) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

DIAGNOSTICS

internal name list overflow
 if the master table contains more than an internally specified number of entries for use by *nlist*(3C).

NAME

tabs − set tabs on a terminal

SYNOPSIS

tabs [tabspec] [**−T**type] [**+m**n]

DESCRIPTION

The *tabs* command 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.

tabspec Four types of tab specification are accepted for *tabspec*. They are described below: canned (−*code*), repetitive (−*n*), arbitrary (*n1,n2,...*), and file (−−*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* Use one of the codes listed below to select a *canned* set of 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 [see *fspec*(4)]:

<: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 [see *fspec*(4)]:

<: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

−n A *repetitive* specification requests tabs at columns 1+*n*, 1+2**n*, etc. 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. 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 formats **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 [see *fspec*(4)]. 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 also may be used; if a given flag occurs more than once, the last value given takes effect:

−T*type* *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* uses the value of the environment variable **TERM**. If **TERM** is not defined in the *environment* [see *environ*(5)], *tabs* tries a sequence that will work for many terminals.

+m*n* 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.

EXAMPLES

tabs −a example using *−code* (*canned* specification) to set tabs to the settings required by the IBM assembler: columns 1, 10, 16, 36, 72.

tabs −8 example of using *−n* (*repetitive* specification), where *n* is **8**, causes tabs to be set every eighth position: 1+(1*8), 1+(2*8), ... which evaluate to columns 9, 17, ...

tabs 1,8,36 example of using *n1,n2,...* (*arbitrary* specification) to set tabs at columns 1, 8, and 36.

tabs --$HOME/fspec.list/att4425
example of using --*file* (*file* specification) to indicate that tabs should be set according to the first line of **$HOME/fspec.list/att4425** [see *fspec*(4)].

DIAGNOSTICS

illegal tabs	when arbitrary tabs are ordered incorrectly
illegal increment	when a zero or missing increment is found in an arbitrary specification
unknown tab code	when a *canned* code cannot be found
can't open	if --*file* option used and file can't be opened
file indirection	if --*file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted

SEE ALSO

newform(1), pr(1), tput(1).
fspec(4), terminfo(4), environ(5), term(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NOTE

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

WARNING

The *tabspec* used with the *tabs* command is different from the one used with the *newform*(1) command. For example, **tabs -8** sets every eighth position; whereas **newform -i-8** indicates that tabs are set every eighth position.

NAME

tail − display the last part of a file

SYNOPSIS

tail [±[number][**lbc**[**f**]]] [file]

DESCRIPTION

The *tail* command 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(1M).

BUGS

Tails relative to the end of the file are stored in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

WARNING

The *tail* command will only tail the last 4096 bytes of a file regardless of its line count.

NAME

tar — file archiver

SYNOPSIS

tar [key] [files]

DESCRIPTION

tar saves and restores files to and from an archive medium, which is typically a storage device such as a floppy disk, a tape, or a regular file. 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. Valid function letters are **c**, **t**, **x**, **u**, and **r**. Other arguments to the command are *files* (or directory names) specifying which files are to be backed up 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 to the end of the archive.

x The named *files* are extracted from the archive. If a named file matches a directory whose contents have been written onto the archive, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *files* argument is given, the contents of the archive are extracted. Note that if several files with the same name are on the archive, the last file overwrites all earlier ones.

t The names of the specified files are listed each time that they occur on the archive. If no *files* argument is given, all the names on the archive are listed.

u The named *files* are added to the archive if they are not already there, or if they have been modified since last written on that archive.

c Creates a new archive; writing begins at the beginning of the archive, instead of after the last file.

e Prevents files from being split across volumes (tapes or floppy disks). If there is not enough room on the present volume for a given file, *tar* prompts for a new volume. This is only valid when the **−k** option is also specified on the command line.

The following characters may be used in addition to the letter that selects the desired function:

0,...,7 This modifier selects the drive on which the archive is mounted. The default drive is 1.

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 archive entries than just the name.

o Causes the extracted files to assume the owner and group ID of the user running the program rather than those on the archive tape.

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 the default device in **/etc/tar/default**. If the name of the file is a dash (−), *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 archive records. The default is 2, the maximum is 20. This option should only be used with raw magnetic tape archives (see **f** above). The number of bytes in a block is BSIZE as defined in **/usr/include/sys/param.h**.

F Causes *tar* to use the next argument as the name of a file from which succeeding arguments are taken. The dash (−) is not a valid argument here.

l Tells *tar* to print an error message if it cannot resolve all of the links to the files being backed up. 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.

k Causes *tar* to use the next argument as the size of an archive volume in kilobytes. The minimum value allowed is 250. This option is useful when the archive is not intended for a magnetic tape device, but for some fixed size device, such as floppy disk (See **f** above). Very large files are split into "extents" across volumes. When restoring from a multivolume archive, *tar* only prompts for a new volume if a split file has been partially restored.

n Indicates the archive device is not a magnetic tape. The **k** option implies this. Listing and extracting the contents of an archive are sped because *tar* can seek over files it wishes to skip. Sizes are printed in kilobytes instead of tape blocks.

p Indicates that files are extracted using their original permissions. It is possible that a non-super-user may be unable to extract files because of the permissions associated with the files or directories being extracted.

EXAMPLES

If the name of a floppy disk device is **/dev/dsk/f1q15dt**, a *tar* format file can be created on this device by typing

tar cvfk /dev/dsk/f1q15dt 360 files

where *files* are the names of files you want archived and 360 is the capacity of the floppy disk in kilobytes. Note that arguments to key letters are given in the same order as the key letters themselves; thus, the **fk** key letters have corresponding arguments, **/dev/dsk/f1q15dt** and **360**. Note that if a *file* is a directory, the contents of the directory are archived recursively. To print a listing of the archive, type

tar tvf /dev/dsk/f1q15dt

At some later time you may want to extract the files from the floppy archive. You can do this by typing

tar xvf /dev/dsk/f1q15dt

The above command extracts all files from the archive using the same pathnames as those of the original archive. Because of this behavior, it is best to save archive files with relative pathnames rather than absolute ones since directory permissions may not let you read the files into the specified absolute directories.

In the above examples, the **v** option is used to confirm the reading or writing of archive files on the screen. Also, a normal file could be substituted for the floppy device, **/dev/dsk/f1q15dt**, in the examples.

FILES

/etc/default/tar Default devices

/tmp/tar*

SEE ALSO

cpio(1), ls(1).

DIAGNOSTICS

Prints an error message about bad key characters and archive read/write errors.

Prints an error message if not enough memory is available to hold the link tables.

NOTES

There is no way to ask for the *n*th occurrence of a file.

The **u** option can be slow.

The **b** option should not be used with archives that are going to be updated. If the archive is on a disk file, the **b** option should not be used at all because updating an archive stored on a disk can destroy it. To update (**r** or **u** option) a *tar* archive, do not use raw magtape and do not use the **b** option. This applies both when updating and when first creating an archive.

The limit on filename length is 100 characters.

When archiving a directory that contains subdirectories, *tar* will only access those subdirectories that are within 17 levels of nesting. Subdirectories at higher levels will be ignored after *tar* displays an error message.

Systems with 1K byte file systems cannot specify raw disk devices unless the **b** option is used to specify an even number of blocks. This means that you cannot update a raw-mode disk partition.

Don't do:

tar xfF – –

This would imply taking two things from the standard input at the same time.

NAME
tee − pipe fitting

SYNOPSIS
tee [−i] [−a] [file] ...

DESCRIPTION
The *tee* command transcribes the standard input to the standard output and makes copies in the *files*.

−i ignore interrupts;

−a causes the output to be appended to the *files* rather than overwriting them.

NAME

test – condition evaluation command

SYNOPSIS

test expr [expr]

DESCRIPTION

The *test* command evaluates the expression *expr* and, if its value is true, sets a zero (true) exit status; otherwise, a non-zero (false) exit status is set; *test* also sets a non-zero exit status if there are no arguments. When permissions are tested, the effective user ID of the process is used.

All operators, flags, and brackets (brackets used as shown in the second SYNOPSIS line) must be separate arguments to the *test* command; normally these items are separated by spaces.

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 [*fildes*]	true if the open file whose file descriptor number is *fildes* (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 also that parentheses are meaningful to the shell and, therefore, must be quoted.

SEE ALSO

find(1), sh(1).

WARNING

If you test a file you own (the **−r**, **−w**, or **−x** tests), but the permission tested does not have the *owner* bit set, a non-zero (false) exit status will be returned even though the file may have the *group* or *other* bit set for that permission. The correct exit status will be set if you are super-user.

The **=** and **!=** operators have a higher precedence than the **−r** through **−n** operators, and **=** and **!=** always expect arguments; therefore, **=** and **!=** cannot be used with the **−r** through **−n** operators.

If more than one argument follows the **−r** through **−n** operators, only the first argument is examined; the others are ignored, unless a **−a** or a **−o** is the second argument.

NAME

tic − terminfo compiler

SYNOPSIS

tic [−v[n]] [−c] file

DESCRIPTION

The *tic* command translates a *terminfo*(4) file from the source format into the compiled format. The results are placed in the directory **/usr/lib/terminfo**. The compiled format is necessary for use with the library routines described in *curses*(3X).

−vn (verbose) output to standard error trace information showing *tic*'s progress. The optional integer *n* is a number from 1 to 10, inclusive, indicating the desired level of detail of information. If *n* is omitted, the default level is 1. If *n* is specified and greater than 1, the level of detail is increased.

−c only check *file* for errors. Errors in **use=** links are not detected.

file contains one or more *terminfo*(4) terminal descriptions in source format [see *terminfo*(4)]. Each description in the file describes the capabilities of a particular terminal. When a **use=***entry-name* field is discovered in a terminal entry currently being compiled, *tic* reads in the binary from **/usr/lib/terminfo** to complete the entry. (Entries created from *file* will be used first. If the environment variable **TERMINFO** is set, that directory is searched instead of **/usr/lib/terminfo**.) *tic* duplicates the capabilities in *entry-name* for the current entry, with the exception of those capabilities that explicitly are defined in the current entry.

If the environment variable **TERMINFO** is set, the compiled results are placed there instead of **/usr/lib/terminfo**.

FILES

/usr/lib/terminfo/?/* compiled terminal description data base

SEE ALSO

curses(3X), term(4), terminfo(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.
Chapter 10 in the *Programmer's Guide*.

WARNINGS

Total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

Terminal names exceeding 14 characters will be truncated to 14 characters and a warning message will be printed.

When the **−c** option is used, duplicate terminal names will not be diagnosed; however, when **−c** is not used, they will be.

BUGS

To allow existing executables from the previous release of the UNIX system to continue to run with the compiled terminfo entries created by the new terminfo compiler, cancelled capabilities will not be marked as cancelled within the terminfo binary unless the entry name has a '+' within it. (Such terminal names are only used for inclusion within other entries via a **use=** entry. Such names would not be used for real terminal names.)

For example:

```
4415+nl, kf1@, kf2@, ....

4415+base, kf1=\EOc, kf2=\EOd, ....

4415-nl|4415 terminal without keys,
    use=4415+nl, use=4415+base,
```

The above example works as expected; the definitions for the keys do not show up in the *4415—nl* entry. However, if the entry *4415+nl* did not have a plus sign within its name, the cancellations would not be marked within the compiled file and the definitions for the function keys would not be cancelled within *4415—nl*.

DIAGNOSTICS

Most diagnostic messages produced by *tic* during the compilation of the source file are preceded with the approximate line number and the name of the terminal currently being worked on.

mkdir ... returned bad status
: The named directory could not be created.

File does not start with terminal names in column one
: The first thing seen in the file, after comments, must be the list of terminal names.

Token after a lseek(2) not NAMES
: Somehow the file being compiled changed during the compilation.

Not enough memory for use_list element
or
Out of memory
: Not enough free memory was available [*malloc*(3C) failed].

Can't open ...
: The named file could not be created.

Error in writing ...
: The named file could not be written to.

Can't link ... to ...
: A link failed.

Error in re-reading compiled file ...
: The compiled file could not be read back in.

Premature EOF
: The current entry ended prematurely.

Backspaced off beginning of line
This error indicates something wrong happened within *tic*.

Unknown Capability - "..."
The named invalid capability was found within the file.

Wrong type used for capability "..."
For example, a string capability was given a numeric value.

Unknown token type
Tokens must be followed by '@' to cancel, ',' for Booleans, '#' for numbers, or '=' for strings.

"...": *bad term name*
or
Line ...: Illegal terminal name - "..."
Terminal names must start with a letter or digit
The given name was invalid. Names must not contain white space or slashes, and must begin with a letter or digit.

"...": *terminal name too long.*
An extremely long terminal name was found.

"...": *terminal name too short.*
A one-letter name was found.

"..." *filename too long, truncating to* "..."
The given name was truncated to 14 characters due to UNIX system file name length limitations.

"..." *defined in more than one entry. Entry being used is* "...".
An entry was found more than once.

Terminal name "..." *synonym for itself*
A name was listed twice in the list of synonyms.

At least one synonym should begin with a letter.
At least one of the names of the terminal should begin with a letter.

Illegal character - "..."
The given invalid character was found in the input file.

New-line in middle of terminal name
The trailing comma was probably left off the list of names.

Missing comma
A comma was missing.

Missing numeric value
The number was missing after a numeric capability.

NULL string value
The proper way to say that a string capability does not exist is to cancel it.

Very long string found. Missing comma?
self-explanatory

Unknown option. Usage is:
An invalid option was entered.

Too many file names. Usage is:
self-explanatory

"..." *nonexistent or permission denied*
The given directory could not be written into.

"..." *is not a directory*
self-explanatory

"...": *Permission denied*
access denied.

"...": *Not a directory*
tic wanted to use the given name as a directory, but it already exists as a file

SYSTEM ERROR!! Fork failed!!!
A *fork*(2) failed.

Error in following up use-links. Either there is a loop in the links or they reference nonexistent terminals. The following is a list of the entries involved:
A *terminfo*(4) entry with a **use**=*name* capability either referenced a nonexistent terminal called *name* or *name* somehow referred back to the given entry.

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

times(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

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. This option works only if the process accounting software is installed. Suboptions **f**, **h**, **k**, **m**, **r**, and **t** modify the data items reported. The options are as follows:

- **−f** Print the *fork/exec* flag and system exit status columns in the output.
- **−h** Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:

  (total CPU time)/(elapsed time).
- **−k** Instead of memory size, show total kcore-minutes.
- **−m** Show mean core size (the default).
- **−r** Show CPU factor:
  (user time/(system-time + user-time)).
- **−t** Show separate system and user CPU times. 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. This option works only if the process accounting software is installed.

−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

sar(1).

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
```

NAME

touch — update access and modification times of a file

SYNOPSIS

touch **[−amc]** [mmddhhmm[yy]] files

DESCRIPTION

The *touch* command 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 *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

tplot − graphics filters

SYNOPSIS

tplot [**−T**terminal [**−e** raster]]

DESCRIPTION

This command reads plotting instructions [see *plot*(4)] from the standard input and in general produce, on the standard output, plotting instructions suitable for a particular *terminal*. If no *terminal* is specified, the environment parameter **$TERM** [see *environ*(5)] is used. Known *terminal*s are:

300	DASI 300.
300S	DASI 300s.
450	DASI 450.
4014	Tektronix 4014.
ver	VERSATEC D1200A. This version of *plot* places a scan-converted image in **/usr/tmp/raster**$$ and sends the result directly to the plotter device, rather than to the standard output. The **−e** option causes a previously scan-converted file *raster* to be sent to the plotter.

FILES

/usr/lib/t300
/usr/lib/t300s
/usr/lib/t450
/usr/lib/t4014
/usr/lib/vplot
/usr/tmp/raster$$

SEE ALSO

plot(3X), plot(4), term(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

tput — initialize a terminal or query terminfo data base

SYNOPSIS

tput [−Ttype] [−S] capname [parms ...]

tput [−Ttype] [−S] **init**

tput [−Ttype] [−S] **reset**

tput [−Ttype] [−S] **longname**

DESCRIPTION

The *tput* command uses the *terminfo*(4) data base to make the values of terminal-dependent capabilities and information available to the shell [see *sh*(1)], to initialize or reset the terminal, or return the long name of the requested terminal type. *tput* outputs a string if the attribute (*cap*ability *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 if the terminal has the capability, **1** for FALSE if it does not), and produces no output. Before using a value returned on standard output, the user should test the exit code [$?, see *sh*(1)] to be sure it is **0**. (See **EXIT CODES** and **DIAGNOSTICS** below.) For a complete list of capabilities and the *capname* associated with each, see *terminfo*(4).

−T*type* indicates the *type* of terminal. Normally, this option is unnecessary because the default is taken from the environment variable **TERM.** If **−T** is specified, then the shell variables **LINES** and **COLUMNS** and the layer size [see *layers*(1)] will not be referenced.

−S causes the *capname* to be read in from standard input instead of from the command line.

capname indicates the attribute from the *terminfo*(4) data base.

parms If the attribute is a string that takes parameters, the arguments *parms* will be inserted into the string. An all numeric argument will be passed to the attribute as a number.

init If the *terminfo*(4) data base is present and an entry for the user's terminal exists (see −T*type,* above), the following will occur: (1) if present, the terminal's initialization strings will be output (**is1**, **is2**, **is3**, **if**, **iprog**), (2) any delays (e.g., new-line) specified in the entry will be set in the tty driver, (3) tabs expansion will be turned on or off according to the specification in the entry, and (4) if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will be silently skipped.

reset Instead of putting out initialization strings, the terminal's reset strings will be output, if present (**rs1**, **rs2**, **rs3**, **rf**). If the reset strings are not present, but initialization strings are, the initialization strings will be output. Otherwise, **reset** acts identically to **init**.

longname If the *terminfo*(4) data base is present and an entry for the user's terminal exists (see **−T***type* above), then the long name of the terminal will be output. The long name is the last name in the first line of the terminal's description in the *terminfo*(4) data base [see *term*(5)].

EXAMPLES

tput init Initialize the terminal according to the type of terminal in the environmental variable **TERM**. This command should be included in everyone's **.profile** after the environmental variable **TERM** has been exported, as illustrated on the *profile*(4) manual page.

tput −T5620 reset Reset an AT&T 5620 terminal, overriding the type of terminal in the environmental variable **TERM**.

tput cup 0 0 Send the sequence to move the cursor to row **0**, column **0** (the upper left corner of the screen, usually known as the "home" cursor position).

tput clear Echo the 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‘

offbold=‘tput rmso‘ Set the shell variables **bold** to begin stand-out mode sequence, and **offbold** to end stand-out mode sequence, for the current terminal. This might be followed by a prompt:

echo "${bold}Please type in your name: ${offbold}\c"

tput hc Set exit code to indicate if the current terminal is a hardcopy terminal.

tput cup 23 4 Send the sequence to move the cursor to row 23, column 4.

tput longname Print the long name from the *terminfo*(4) data base for the type of terminal specified in the environmental variable **TERM**.

FILES

/usr/lib/terminfo/?/*	compiled terminal description data base
/usr/include/curses.h	*curses*(3X) header file
/usr/include/term.h	*terminfo*(4) header file
/usr/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs); for more information, see the "Tabs and Initialization" section of *terminfo*(4)

SEE ALSO

stty (1), tabs (1).

profile(4), terminfo(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

Chapter 10 of the *Programmer's Guide*.

EXIT CODES

If *capname* is of type Boolean, a value of **0** is set for TRUE and **1** for FALSE.

If *capname* is of type string, a value of **0** is set if the *capname* is defined for this terminal *type* (the value of *capname* is returned on standard output); a value of **1** is set if *capname* is not defined for this terminal *type* (a null value is returned on standard output).

If *capname* is of type integer, a value of **0** is always set, whether or not *capname* is defined for this terminal *type*. To determine if *capname* is defined for this terminal *type*, the user must test the value of standard output. A value of **−1** means that *capname* is not defined for this terminal *type*.

Any other exit code indicates an error; see **DIAGNOSTICS**, below.

DIAGNOSTICS

tput prints the following error messages and sets the corresponding exit codes:

exit code	error message
0	−1 (*capname* is a numeric variable that is not specified in the *terminfo*(4) data base for this terminal *type*, e.g., **tput −T450 lines** and **tput −T2621 xmc**)
1	no error message is printed, see **EXIT CODES** above.
2	usage error
3	unknown terminal *type* or no *terminfo*(4) data base
4	unknown *terminfo*(4) capability *capname*

NAME

tr − translate characters

SYNOPSIS

tr [**−cds**] [string1 [string2]]

DESCRIPTION

The *tr* command 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.

EXAMPLE

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 alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for new-line.

```
tr −cs "[A−Z][a−z]" "[\012*]" <file1 >file2
```

SEE ALSO

ed(1), sh(1).
ascii(5) in the *INTERACTIVE SDS Guide and Programmer's Reference 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.

NAME

tset – provide information to set terminal modes

SYNOPSIS

tset [*options*] [*type*]

DESCRIPTION

tset allows the user to set a terminal's ERASE and KILL characters, and define the terminal's type and capabilities by creating values for the TERM environment variable. *tset* initializes or resets the terminal with *tput*(1). If a *type* is given with the **–s** option, *tset* creates information for a terminal of the specified type. The type may be any type given in the *terminfo* database. If the *type* is not specified with the **–s** option, *tset* creates information for a terminal of the type defined by the value of the environment variable, TERM, unless the **–h** or **–m** option is given. If the **TERM** variable is defined, *tset* uses the *terminfo* database entry. If these options are used, *tset* searches the **/etc/ttytype** file for the terminal type corresponding to the current serial port; it then creates information for a terminal based on this type. If the serial port is not found in **/etc/ttytype**, the terminal type is set to **unknown**.

tset displays the created information at the standard output. The information is in a form that can be used to set the current environment variables. The exact form depends on the login shell from which *tset* was invoked. The following examples illustrate how to use this information to change the variables.

There are the following options:

–e[*c*] Sets the ERASE character to *c* on all terminals. The default setting is the BACKSPACE, or CTRL-H.

–E[*c*] Identical to the **–e** command except that it only operates on terminals that can BACKSPACE.

–k[*c*] Sets the KILL character to *c*, defaulting to CTRL-U.

– Prints the terminal type on the standard output.

–s Outputs the "setenv" commands [for *csh*(1)], or "export" and assignment commands [for *sh*(1)]. The type of commands are determined by the user's login shell.

–h Forces *tset* to search **/etc/ttytype** for information and to overlook the environment variable, TERM.

–S Only outputs the strings to be placed in the environment variables, without the shell commands printed for **-S**.

–r Prints the terminal type on the diagnostic output.

–Q Suppresses the printing of the "Erase set to" and "Kill set to" messages.

–I Suppresses printing of the terminal initialization strings, e.g., spawns *tput reset* instead of *tput init.*

–m[*ident*][*test baudrate*]:*type*
Allows a user to specify how a given serial port is is to be mapped to an actual terminal type. The option applies to any serial port in **/etc/ttytype** whose type is indeterminate (e.g.,

dialup, *plugboard*, etc.). The *type* specifies the terminal type to be used, and *ident* identifies the name of the indeterminate type to be matched. If no *ident* is given, all indeterminate types are matched. The *test baudrate* defines a test to be performed on the serial port before the type is assigned. The *baudrate* must be as defined in *stty*(1). The *test* may be any combination of: >, =, <, @, and !. If the *type* begins with a question mark, the user is asked if he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. The question mark must be escaped to prevent filename expansion by the shell. If more than one **−m** option is given, the first correct mapping prevails.

tset is most useful when included in the **.login** [for *csh*(1)] or **.profile** [for *sh*(1)] file executed automatically at login, with **−m** mapping used to specify the terminal type you most frequently dial in on.

EXAMPLES

tset gt42

tset −mdialup\>300:adm3a −mdialup:dw2 −Qr −e#

tset −m dial:ti733 −m plug:\?hp2621 −m unknown:\? −e −k^U

To use the information created by the **−s** option for the Bourne shell, (*sh*), repeat these commands:

```
tset −s ... > /tmp/tset$$
/tmp/tset$$
rm /tmp/tset$$
```

To use the information created for *csh*, use:

```
set noglob
set term=(‘tset −S ....‘)
setenv TERM $term[1]
unset term
unset noglob
```

FILES

/etc/ttytype	Port name to terminal type map database
/usr/lib/terminfo/*	Terminal capability database

SEE ALSO

stty(1), termio(7), tput(1), tty(1).

terminfo(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NOTES

This utility was developed at the University of California at Berkeley and is used with permission.

NAME

tty — get the name of the terminal

SYNOPSIS

tty [−l] [−s]

DESCRIPTION

The *tty* command prints the path name of the user's terminal.

−l prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line.

−s 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.

NAME

ttymap — set terminal mapping and scancode translation

SYNOPSIS

ttymap mapfile

ttymap −r

ttymap −d

DESCRIPTION

ttymap is a utility that permits a user to activate character mapping on input and output for the user's terminal. This same utility can be used for regular terminals as well as for scancode devices such as the AT console. It makes full use of all the features of the terminal (tty) driver and the keyboard display driver that support such mapping.

The command *ttymap mapfile* reads the contents of the file *mapfile* and sets the corresponding mapping as supported by the terminal driver and/or keyboard/display driver. The layout of the *mapfile* and the functionality supported by both drivers are described below.

ttymap −d disables the current mapping by the terminal driver.

ttymap −r resets the scancode translation back to that of a U.S. PC keyboard.

Terminal Mapping

The original UNIX operating system was written to support the ASCII codeset. ASCII is one of many standards to represent a number of characters internally as certain numbers. Typical for ASCII is that it supports 128 different characters, each represented by a single byte of which the 8^{th} bit is not used. Many UNIX system applications, including the shell, took advantage of this. Starting with UNIX System V Release 3.1, most of these applications have been modified to properly support characters represented as a byte with the 8^{th} bit set as well. This means that now 256 characters can be supported at the same time. However, a consistent coding convention needs to be applied. In the IBM PC world, an 8-bit coding referred to as IBM extended ASCII has been used for several years; MS-DOS users are quite familiar with that. In heterogeneous UNIX System environments, a different codeset, called ISO8859, has been promoted. In both codesets, characters found in the ASCII codeset are represented in the same way. The other 128 characters are encoded differently, however, and some characters found in one codeset will be missing in the other. The INTERACTIVE UNIX Operating System supports both codesets; actually, it supports any 8-bit one byte codeset.

To be able to use characters from the French, German, Finnish, and other alphabets, several terminals are available on the market that generate 7-bit codes but display the above-mentioned characters on the screen instead of the ones found on a U.S. terminal. On the keyboard there are an equal number of keys, but there are different characters on the key caps. Others, such as a DEC VT220, will support 256 different characters at a time but use their own proprietary codesets.

Assume you are using the INTERACTIVE UNIX Operating System with a console and a French 7-bit terminal connected to the serial port. If you edit a file on the terminal and use the French character *é* in text, the terminal will actually generate the ASCII code 123, which is the code normally used for the left curly brace. If you look at the edited file on the console, the letter will actually appear to be a curly brace. Therefore, input and output mapping should be supported by the terminal driver to allow the consistent use of one single codeset throughout the system. The INTERACTIVE UNIX Operating System supports all mapping features that are now standard in the System V Release 3.2 terminal driver, as well as some enhancements by INTERACTIVE Systems Corporation.

Input mapping
On input, any byte can be mapped to any byte. Using the example above, you could map 123 to 130, the code used for *é* in the IBM extended ASCII codeset.

Output mapping
On output, any byte can be mapped to either a byte or a string. In the above example, 130 would be mapped back to 123 to properly display the character on the screen. If the connected device is a printer that does not support the *é* character, it could be mapped to the string:

e BACKSPACE '

Dead keys
On typewriters, keys can be found that behave slightly differently than all the others, because when you press them, the printing wheel of the typewriter does not move. **CTRL** (^) is such a character. When it is followed by an *e*, the letter *e* is generated. This is called a deadkey or a non-spacing character. The terminal driver supports the use of deadkeys. Typically, the ^ character and the umlaut character are used as deadkeys.

Compose sequences
Characters can also be generated using a compose sequence. A dedicated character called the "compose character" followed by two other keystrokes will generate a single character. As an example, COMPOSE followed by the plus and the minus sign could generate the plus/minus sign (±). Compose sequences can also be used as an alternative for deadkeys, e.g., "COMPOSE ^ e" instead of "^ e."

Decimal representation
Rarely used characters can be generated by pressing the compose key followed by three digits.

Toggle key
An optional toggle key can be defined to temporarily disable the current mapping from within an application. This can be useful when, for example, a German programmer wants easy access to the curly braces and the brackets.

Scancode Mapping

The keyboards of the console and some other peripherals such as SunRiver workstations behave differently than those of regular terminals. They generate what are called *scancodes* and you will also find a number of keys on these keyboards, such as the ALT key, that are not found on regular terminals. Scancodes generated by PC keyboards typically represent the location of the key on the keyboard. The keyboard driver has to properly translate these scancodes. The different national variants of a PC keyboard not only have non-English characters printed on some of the keycaps, but the order of some of the keys is different as well. Without changing the scancode translation, a French user would type **A** and see a **Q** on his screen. Several status keys can influence the translated code as well. The keyboard driver, and thus the *ttymap* program, makes a distinction between two sets of key combinations that can be translated.

Function keys
Up to 60 key combinations are recognized as function keys. The first 12 are the 12 function keys of a 101-key PC-keyboard (the first 10 on an 84-key keyboard).

If you do not know whether you have an 84- or 101-key keyboard, you can use the following scheme to determine which type you have:

> If your keyboard has arrow keys that are separate from the ones on the numeric keypad, then you have a 101-key keyboard.
>
> If the arrow keys on your keyboard are located on the numeric keypad only, then you have an 84-key keyboard.

F13 to F24 are the same keys used in combination with **SHIFT**, F25 to F36 when used with **CTRL**, and F37 to F48 when used with **CTRL** and **SHIFT** together. F49 to F60 are the keys on the numeric keypad, in the following order:

7
8
9
–
4
5
6
+
1
2
3
INS

Each of these function keys can be given a string as a value. The total length of all strings should not exceed 512 characters. See *keyboard*(7) for a list of default values.

Regular keys
Scancodes generated by all keys on the PC keyboard can be translated in a different way as well. For each key, a different translation can be specified for each of the following four cases:

1. The key is pressed.
2. The key and the **SHIFT** key are pressed simultaneously.
3. The key and the **ALT** key are pressed simultaneously
4. The key, the **SHIFT**, and the **ALT** keys are pressed simultaneously

For each of these cases, the scancode can be translated into one of the following:

a single byte
a single byte preceded by ESC N
a single byte preceded by ESC O
a single byte preceded by ESC [

Internally, special bits are set to indicate that an escape sequence needs to be generated. Other bits are used to indicate whether the translated code should be influenced by some special keys.

NUM LOCK
If the **NUM LOCK** bit is set, the regular and **SHIFT** values are swapped, as are the **ALT** and **SHIFT ALT** values, whenever the **NUM LOCK LED** is on. By default, only the keys on the numeric keypad have this bit set. That is why these keys generate 7, 8, 9, etc. when the **NUM LOCK LED** is on, which is the same value that would be produced if **SHIFT** were used with these keys.

CAPS LOCK
This has the same effect as the **NUM LOCK** key. By default, this bit is set for all letters and not set for punctuation signs.

CTRL When a key is translated into a single byte (no escape sequence) and this bit is set, the corresponding control character will be generated when the **CTRL** key is pressed simultaneously. This is equally valid for the **SHIFT**, **ALT**, and **SHIFT ALT** combination. When this bit is not used, the **CTRL** key combination will not generate anything.

mapfiles

This section describes the layout of a *mapfile* that is read by the *ttymap* program.

A *mapfile* is a text file that consists of several **sections**. A sharp sign (**#**) can be used to include comments. Everything following the **#** until the end of the line will be ignored by the *ttymap* program. Inside a line, C-style comments can be used as well. The beginning of each section is indicated by a *keyword*. Spaces and tabs are silently ignored and can be used at all times to improve readability. All but one section, the one that defines the *compose character*, can be left out. The order in which the different sections should appear is predefined. Here is the list of keywords in the order they should appear:

input:
toggle:
dead:
compose:
output:
scancodes:

Characters can be described in several different ways. ASCII characters can be described by putting them between single quotes. For example:

'a' '{'

Between single quotes, control characters can be listed by using a circumflex sign before the character that needs to be quoted. For example:

```
'^x'
```

When a backslash (\) is used, what follows will be interpreted as a decimal, octal (leading zero), or hexadecimal (leading x or X) representation of the character, although in this case the use of single quotes is not mandatory. For example:

```
'\x88'
```

is the same as:

```
0x88  (zero needed when not quoted)
```

and:

```
'\007'
```

is the same as:

```
007
```

When strings are needed, a list of character representations should be used. Quoted strings will be supported in the future.

The following paragraphs describe what goes in each section.

Input section
The input section describes which input characters should be mapped into a single byte. A very small sample input section could be:

```
input:
'A' 'B'         # map A into B on input
'#' 0x9c        # map sharp sign into pound sign
```

Toggle section
The toggle section is a one-line section that defines which key is to toggle between mapping and no mapping. For example:

```
toggle:
'^y'            # ctrl y is the toggle key
```

Deadkey section
The deadkey section defines which keys should be treated as deadkeys. A **dead:** keyword followed by the specification of the character appears in this section for each deadkey. The subsequent lines describe what key should be generated for each key following the deadkey. A deadkey followed by a key not described in this part of the *mapfile* will not generate any key and a beep tone will be produced on the terminal. For example:

```
dead: '^'       # circumflex is a deadkey
' '   '^'       # circumflex followed by space generates circumflex
'e'   0x88      # circumflex followed by e generates e circumflex
dead: '"'       # double quote used as a deadkey
' '   '"'       # double quote space generates double quote
'a'   0x84      # double quote a generates an umlaut
```

Compose section
The first line of this section describes what the compose character is.

That line should always be present in the *mapfile*. Subsequent lines consist of three character representations indicating each time that the third character needs to be generated on input when the compose character is followed by the first two. Compose sequences with the same first character should be grouped together. For example:

```
compose: '^x'
'"' 'e'   0x89  # e with umlaut is generated when typing ^x " e
'"' 'a'   0x84  # a with umlaut
'e' '"'   0x89  # e with umlaut is generated when typing ^x e "
'a' '"'   0x84  # a with umlaut
```

The following example would give the wrong result. All lines starting with the same character specification should be grouped together.

```
compose: '^x'
'"' 'e'   0x89  # e with umlaut is generated when typing ^x " e
'e' '"'   0x89  # e with umlaut is generated when typing ^x e "
'"' 'a'   0x84  # a with umlaut
'a' '"'   0x84  # a with umlaut
```

Output section
This section describes the mapping on output, either single byte to single byte, or single byte to string. A string is specified as a series of character specifications. For example:

```
output:

0x82 '{'              # map e with accent to { to display e with accent
'^u' '("K"I"L"L")'    # print (KILL) when kill character is used
```

Scancodes section
This section will only have an effect when your terminal is a scancode device. No error message will be produced when this section is mistakenly in your *mapfile*, because the *ttymap* program will find out whether the terminal is a scancode device or not. The lines in this section can have two different formats. One format will be used to describe what the values of the function keys must be. The other format describes the translation of scancodes into a byte or an escape sequence. No specific order is required.

Function keys
Here is an example of a line defining a string for a function key:

```
F13  'd"a"t"e"\n'   # SHIFT F1 is the date command
```

The numbering convention of the functionkeys is described in a previous section. Currently, the use of quoted strings such as "*date\n*" is not supported.

Scancodes
Specifying how to translate a scancode is a more complex task. The general format of such a line is:

```
scancode normal shift alt shiftalt flags
```

scancode should list the hexadecimal representation of a scancode generated by a key (unquoted). How keys correspond with scancodes can be found in *keyboard*(7).

normal, shift, alt and **shiftalt** are character representations in one of the formats described throughout this document, optionally followed by one of the following special keywords:

|C This indicates that the key is influenced by the **CTRL** key.

|N This indicates that **ESC N** should preceed the specified character.

|O This indicates that **ESC O** should preceed the specified character.

|[This indicates that **ESC [** should preceed the specified character.

The **normal** field defines how the scancode is translated when no other key is pressed, the **shift** field defines the translation for when the **SHIFT** key is used simultaneously, the **alt** field specifies what to do when the **ALT** key is pressed together with this and the **shiftalt** field contains the information on what to generate when both the **SHIFT** and **ALT** keys are pressed.

All five fields must be filled in. When no translation is requested (that is, the current active translation does not need to be changed) a dash (−) can be used. The sixth field is optional. This field can contain the special keyword **CAPS** or **NUM** or both, to indicate whether or not the **CAPS LOCK** key or **NUM LOCK** key status have any effect. Here is a sample line that describes the default translation for the **'Q'** key:

```
0x10 'q'|C  'Q'|C  'q'|N  'Q'|N  CAPS
```

If the normal or shift field is filled out for a scancode that represents a function key, a self-explanatory message will be produced and that translation information will be ignored.

A more detailed example of a **scancodes** section is:

```
scancodes:
# the w key
0x11 'w'|C  'W'|C  'w'|N  'W'|N  CAPS
# left square bracket and curly brace key
# control shift [ does not generate anything (no C flag)
0x1a '['|C  '{'    '['|N  '{'|N
# 9 on numeric keypad
0x49 'V'|[  '9'    '9'|N  '9'|N  NUM
F13 'd''a''t''e''0 # SHIFT F1
```

More complete examples of *mapfile*s can be found in **/usr/lib/keyboard/usa.map** and **/usr/lib/keyboard/*.map**.

FILES

/usr/lib/keyboard/usa.map	sample *mapfile* for using compose character sequences and deadkeys on a U.S. keyboard
/usr/lib/keyboard/*.map	sample *mapfile*s for European keyboards without compose and deadkey sections
/usr/lib/keyboard/keys	dump of default keytable for PC keyboard
/usr/lib/keyboard/strings	dump of default stringtable for PC keyboard

SEE ALSO

stty(1), keyboard(7), termio(7).

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

Uutry — try to contact remote system with debugging on

SYNOPSIS

/usr/lib/uucp/Uutry [**−x** debug_level] [**−r**] system_name

DESCRIPTION

Uutry is a shell that is used to invoke *uucico* to call a remote site. Debugging is turned on (default is level 5); **−x** will override that value. The **−r** overrides the retry time in **/usr/spool/uucp/.status**. The debugging output is put in file **/tmp/***system_name.* A tail **−f** of the output is executed. A <DELETE> or <BREAK> will give control back to the terminal while the *uucico* continues to run, putting its output in **/tmp/***system_name.*

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*
/tmp/system_name

SEE ALSO

uucico(1M), uucp(1C), uux(1C).

NAME

uadmin – administrative control

SYNOPSIS

/etc/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 *cmd* (command) and *fcn* (function) are converted to integers and passed to the *uadmin* system call.

SEE ALSO

uadmin(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

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.

The *umask* command is recognized and executed by the shell.

The *umask* command can be included in the user's **.profile** [see *profile*(4)] and invoked at login to automatically set the user's permissions on files or directories created.

SEE ALSO

chmod(1), sh(1).
chmod(2), creat(2), umask(2), profile(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

uname – print name of current UNIX system

SYNOPSIS

uname [–snrvma]
uname [–S system name **]**

DESCRIPTION

The *uname* command 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 *system name* (default).

–n print *nodename* (the nodename is the name by which the system is known 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.

On your computer, the system name and the nodename may be changed by specifying a *system name* argument to the **–S** option. The *system name* argument is restricted to 8 characters. Only the super-user is allowed this capability.

SEE ALSO

uname(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

uniq − report repeated lines in a file

SYNOPSIS

uniq [**−udc** [+n] [−n]] [input [output]]

DESCRIPTION

The *uniq* command 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).

NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

The *units* command 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
```

The *units* command does only 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:

pi	ratio of circumference to diameter,
c	speed of light,
e	charge on an electron,
g	acceleration of gravity,
force	same as **g**,
mole	Avogadro's number,
water	pressure head per unit height of water,
au	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

uucheck – check the uucp directories and permissions file

SYNOPSIS

/usr/lib/uucp/uucheck [**–v**] [**–x** debug_level]

DESCRIPTION

The *uucheck* command checks for the presence of the *uucp* system required files and directories. Within the *uucp* makefile, it is executed before the installation takes place. It also checks for some obvious errors in the Permissions file (**/usr/lib/uucp/Permissions**). When executed with the **–v** option, it gives a detailed explanation of how the uucp programs will interpret the Permissions file. The **–x** option is used for debugging. *debug_level* is a single digit in the range 1-9; the higher the value, the greater the detail.

Note that *uucheck* can only be used by the super-user or *uucp*.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuscheds
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

uucico(1M), uucp(1C), uusched(1M), uustat(1C), uux(1C).

BUGS

The program does not check file/directory modes or some errors in the Permissions file such as duplicate login or machine name.

NAME

uucico − file transport program for the uucp system

SYNOPSIS

/usr/lib/uucp/uucico [**−r** *role_number*] [**−x** *debug_level*] [**−i** *interface*] [**−d** *spool_directory*] **−s** *system_name*

DESCRIPTION

uucico is the file transport program for *uucp* work file transfers. Role numbers for the **−r** are the digit 1 for master mode or 0 for slave mode (default). The **−r** option should be specified as the digit 1 for master mode when *uucico* is started by a program or *cron*. *Uux* and *uucp* both queue jobs that will be transferred by *uucico.* It is normally started by the scheduler, *uuschedfl, but can be started manually; this is done for debugging. For example, the shell Uutry* starts *uucico* with debugging turned on. A single digit must be used for the **−x** option with higher numbers for more debugging.

The **−i** option defines the *interface* used with *uucico.* This interface only affects slave mode. Known interfaces are UNIX (default), TLI (basic Transport Layer Interface), and TLIS (Transport Layer Interface with Streams modules, read/write).

The **−d** option is used to specify the directory (*spool_directory*) that contains the work files to be transferred. The default spool directory is /usr/spool/uucp. The **−s** option defines the system (*system_name*) that *uucico* will try to contact. The *system_name* must be defined in the **Systems** file.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Devconfig
/usr/lib/uucp/Sysfiles
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

uucp(1C), uustat(1C), uux(1C), cron(1M), Uutry(1M), uusched(1M).

NAME

uucleanup – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uucleanup [**–C** time] [**–W** time] [**–X** time] [**–m** string] [**–o** time] [**–s** system]

DESCRIPTION

The *uucleanup* command will scan the spool directories for old files and take appropriate action to remove them in a useful way:

Inform the requestor of send/receive requests for systems that cannot be reached.

Return mail, which cannot be delivered, to the sender.

Delete or execute rnews for rnews type files (depending on where the news originated––locally or remotely).

Remove all other files.

In addition, there is provision to warn users of requests that have been waiting for a given number of days (default 1). Note that *uucleanup* will process as if all option *times* were specified to the default values unless *time* is specifically set.

The following options are available.

–C*time* Any **C.** files greater or equal to *time* days old will be removed with appropriate information to the requestor. (default 7 days)

–D*time* Any **D.** files greater or equal to *time* days old will be removed. An attempt will be made to deliver mail messages and execute rnews when appropriate. (default 7 days)

–W*time* Any **C.** files equal to *time* days old will cause a mail message to be sent to the requestor warning about the delay in contacting the remote. The message includes the *JOBID*, and in the case of mail, the mail message. The administrator may include a message line telling whom to call to check the problem (**–m** option). (default 1 day)

–X*time* Any **X.** files greater or equal to *time* days old will be removed. The **D.** files are probably not present (if they were, the **X.** could get executed). But if there are **D.** files, they will be taken care of by D. processing. (default 2 days)

–m*string* This line will be included in the warning message generated by the **–W** option.

–o*time* Other files whose age is more than *time* days will be deleted. (default 2 days) The default line is "See your local administrator to locate the problem."

–s*system* Execute for *system* spool directory only.

−x*debug_level*

The **−x** debug level is a single digit between 0 and 9; higher numbers give more detailed debugging information. (If *uucleanup* was compiled with -DSMALL, no debugging output will be available.)

This program is typically started by the shell *uudemon.cleanup*, which should be started by *cron*(1M).

FILES

/usr/lib/uucp	directory with commands used by *uucleanup* internally
/usr/spool/uucp	spool directory

SEE ALSO

cron(1M), uucp(1C), uux(1C).

NAME

uucp, uulog, uuname – UNIX-to-UNIX system copy

SYNOPSIS

uucp [options] source-files destination-file
uulog [options] **−s**system
uulog [options] system
uulog [options] **−f**system
uuname [**−l**] [**−c**]

DESCRIPTION

uucp

uucp copies files named by the *source-files* argument 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 that *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, to the destination. See WARNINGS and BUGS below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to forward information (see WARNINGS below for restrictions).

The following shell metacharacters are disallowed in *system-name*:

‘ ; & | ^ < > ( ) <CR> <TAB> <SPACE>

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 ~/*destination* where *destination* is appended to **/usr/spool/uucppublic**; [NOTE: This destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example ~**/dan/** as the destination will make the directory **/usr/spool/uucppublic/dan** if it does not exist and put the requested file(s) in that directory].

(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*:

−c Do not copy local file to the spool directory for transfer to the remote machine (default).

−C Force the copy of local files to the spool directory for transfer.

−d Make all necessary directories for the file copy (default).

−f Do not make intermediate directories for the file copy.

−g*grade* *Grade* is a single letter/number; lower ascii sequence characters will cause the job to be transmitted earlier during a particular conversation.

−j Output the job identification ASCII string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.

−m Send mail to the requester when the copy is completed.

−n*user* Notify *user* on the remote system that a file was sent.

−r Do not start the file transfer, just queue the job.

−s*file* Report status of the transfer to *file*. Note that the *file* must be a full path name.

−x*debug_level*
Produce debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

uulog

uulog queries a log file of *uucp* or *uuxqt* transactions in a file

/usr/spool/uucp/.Log/uucico/*system,*
or
/usr/spool/uucp/.Log/uuxqt/*system.*

The options cause *uulog* to print logging information:

−s*sys* Print information about file transfer work involving system *sys*.

−f*system* Does a "tail **−f**" of the file transfer log for *system*. (You must hit BREAK to exit this function.) Other options used in conjunction with the above:

−x Look in the *uuxqt* log file for the given system.

−*number* Indicates that a "tail" command of *number* lines should be executed.

uuname

uuname lists the names of systems known to *uucp*. The **−c** option returns the names of systems known to *cu*. (The two lists are the same, unless your machine is using different *Systems* files for *cu* and *uucp*. See the *Sysfiles* file.) The **−l** option returns the local system name.

FILES

/usr/spool/uucp	spool directories
/usr/spool/uucppublic/*	public directory for receiving and sending (**/usr/spool/uucppublic**)
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail(1), uustat(1C), Uutry(1M), uux(1C), uuxqt(1M).

chmod(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

WARNINGS

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 ~/).

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 forwarding of files through other systems may not be compatible with the previous version of *uucp*. If forwarding is used, all systems in the route must have the same version of *uucp*.

BUGS

Protected files and files that are in protected directories that are owned by the requester can be sent by *uucp* using the −C option. However, if the requestor is **root**, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

NAME

uugetty − set terminal type, modes, speed, and line discipline

SYNOPSIS

/usr/lib/uucp/uugetty [**−t** *timeout*] [**−r**] *line* [*speed* [*type* [*linedisc*]]]

/usr/lib/uucp/uugetty −c *file*

DESCRIPTION

uugetty is a standard *getty*(1M) modified to allow a tty line to be used by *uucico*, *cu*, and *ct*; that is, the line can be used in both directions. The *uugetty* will allow users to login, but if the line is free, *uucico*, *cu*, or *ct* can use it for dialing out. The implementation depends on the fact that *uucico*, *cu*, and *ct* create lock files when devices are used. When the "open()" returns (or the first character is read when **−r** option is used), the status of the lock file indicates whether the line is being used by *uucico*, *cu*, *ct*, or someone trying to login. Note that in the **−r** case, several <carriage-return> characters may be required before the login message is output. The human users will be able to handle this slight inconvenience. *uucico* trying to login will have to be told by using the following login script:

"" \r\d\r\d\r\d\r in:--in: ...

where the ... is whatever would normally be used for the login sequence.

If there is a *uugetty* on one end of a direct line, there must be a *uugetty* on the other end as well. Here is an **/etc/inittab** entry using *uugetty* on an intelligent modem or direct line:

30:2:respawn:/usr/lib/uucp/uugetty −r −t 60 tty00 1200

The meanings of the available options are:

−t *timeout*
: Specifies that *uugetty* should exit if the open on the line succeeds and there is no response to the login prompt in *timeout* seconds. *timeout* is replaced by an integer.

−r
: Causes *uugetty* to wait to read a character before it puts out the login message, thus preventing two *uugetty*s from looping. An entry for an intelligent modem or direct line that has a *uugetty* on each end must use this option.

line
: Defines the name of the line to which *uugetty* will attach itself. The line name will point to an entry in the **/dev** directory. For example, **/dev/tty00**.

speed
: Defines the entry to use from the **/etc/gettydefs** file. The entry defines the line speed, the login message, the initial tty setting, and the next speed to try if the user says the speed is inappropriate (by sending a *break* character). The default *speed* is 300.

type
: Defines the type of terminal connected to the line. The default terminal is **none**, representing a normal terminal unknown to the system.

linedisc Sets the line discipline to use on the line. The default is **LDISC0**, which is the only one currently compiled into the operating system.

−c *file* Checks the speed and tty definitions in *file* and sends the results to standard output. Unrecognized modes and improperly constructed entries are reported. For correct entries, flag values are printed. *file* is replaced by **/etc/gettydefs** or a similarly structured file.

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

login(1), ct(1C), cu(1C), getty(1M), init(1M), uucico(1M), tty(7).

ioctl(2), gettydefs(4), inittab(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

BUGS

ct will not work when *uugetty* is used with an intelligent modem such as penril or ventel.

NAME

uusched — the scheduler for the uucp file transport program

SYNOPSIS

/usr/lib/uucp/uusched [**−x** debug_level] [**−u** debug_level]

DESCRIPTION

The *uusched* command is the *uucp* file transport scheduler. It is usually started by the daemon *uudemon.hour* that is started by *cron*(1M) from an entry in **/usr/spool/cron/crontab**:

39 * * * * /bin/su uucp -c "/usr/lib/uucp/uudemon.hour > /dev/null"

The two options are for debugging purposes only; **−x** *debug_level* will output debugging messages from *uusched* and **−u** *debug_level* will be passed as **−x** *debug_level* to *uucico*. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), uucico(1M), uucp(1C), uustat(1C), uux(1C).

NAME

uustat — uucp status inquiry and job control

SYNOPSIS

uustat [−a]
uustat [−m]
uustat [−p]
uustat [−q]
uustat [**−k**jobid]
uustat [**−r**jobid]
uustat [**−s** system] [**−u**user]

DESCRIPTION

The *uustat* command will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. Only one of the following options can be specified with *uustat* per command execution:

−a Output all jobs in queue.

−m Report the status of accessibility of all machines.

−p Execute a "ps −flp" for all the process-ids that are in the lock files.

−q List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in () next to the number of **C** or **X** files, it is the age in days of the oldest **C./X.** file for that system. The Retry field represents the number of hours until the next possible call. The Count is the number of failure attempts. NOTE: For systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute. As an example of the output produced by the **−q** option:

eagle	3C	04/07-11:07	NO DEVICES AVAILABLE
mh3bs3	2C	07/07-10:42	SUCCESSFUL

The above output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

−k*jobid* Kill the *uucp* request whose job identification is *jobid*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

−r*jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the daemon.

Either or both of the following options can be specified with *uustat*:

−s*sys* Report the status of all *uucp* requests for remote system *sys*.

−u*user* Report the status of all *uucp* requests issued by *user*.

Output for both the **−s** and **−u** options has the following format:

```
eaglen0000   4/07-11:01:03   (POLL)
eagleN1bd7   4/07-11:07      Seagledan522 /usr/dan/A
eagleC1bd8   4/07-11:07      Seagledan59 D.3b2al2ce4924
             4/07-11:07      Seagledanrmail mike
```

With the above two options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an 'S' or 'R' depending on whether the job is to send or request a file. This is followed by the user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (*rmail* — the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (e.g., D.3b2alce4924) that is created for data files associated with remote executions (*rmail* in this example).

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user.

FILES

/usr/spool/uucp/* spool directories

SEE ALSO

uucp(1C).

NAME

uuto, uupick – public UNIX system to UNIX system file copy

SYNOPSIS

uuto [options] source-files destination
uupick [**–s** system]

DESCRIPTION

The *uuto* command 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*). *User* 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. By default this directory is **/usr/spool/uucppublic**. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by *mail*(1) of the arrival of files.

The *uupick* command 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*] ?

The *uupick* command 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*. If *dir* is not specified as a complete path name (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
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.

The *uupick* command invoked with the −s*system* option will only search the PUBDIR for files sent from *system*.

FILES

PUBDIR/usr/spool/uucppublic public directory

SEE ALSO

mail(1), uucleanup(1M), uucp(1C), uustat(1C), uux(1C).

WARNINGS

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.

NAME

uux — UNIX system to UNIX system command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

The *uux* command 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: For security reasons, most installations limit the list of commands executable on behalf of an incoming request from *uux*, permitting only the receipt of mail [see *mail*(1)]. (Remote execution permissions are defined in **/usr/lib/uucp/Permissions.**)

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/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"

will get the **file1** and **file2** files from the "usg" and "pwba" machines, execute a *diff*(1) command, and put the results in **file.diff** in the local **PUBDIR/dan/** 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.

The *uux* command will attempt to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command

uux a!cut −f1 b!/usr/file \(c!/usr/file\)

gets **/usr/file** from system "b" and sends it to system "a", performs a *cut* command on that file, and sends the result of the *cut* command to system "c".

The *uux* command will notify you if the requested command on the remote system was disallowed. This notification can be turned off by the **−n** option. The response comes by remote mail from the remote machine.

The following *options* are interpreted by *uux*:

−	The standard input to *uux* is made the standard input to the *command-string*.
−a*name*	Use *name* as the user identification replacing the initiator user-id. (Notification will be returned to the user.)
−b	Return whatever standard input was provided to the *uux* command if the exit status is non-zero.
−c	Do not copy local file to the spool directory for transfer to the remote machine (default).
−C	Force the copy of local files to the spool directory for transfer.
−g*grade*	*Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.
−j	Output the jobid ASCII string on the standard output, which is the job identification. This job identification can be used by *uustat* to obtain the status or terminate a job.
−n	Do not notify the user if the command fails.
−p	Same as −: The standard input to *uux* is made the standard input to the *command-string*.
−r	Do not start the file transfer, just queue the job.
−s*file*	Report status of the transfer in *file*.
−x*debug_level*	Produce debugging output on the standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.
−z	Send success notification to the user.

FILES

/usr/lib/uucp/spool	spool directories
/usr/lib/uucp/Permissions	remote execution permissions
/usr/lib/uucp/*	other data and programs

SEE ALSO

cut(1), mail(1), uucp(1C), uustat(1C).

WARNINGS

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.

The execution of commands on remote systems takes place in an execution directory known to the *uucp* system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the *uux* request. The following command will NOT work:

```
uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

but the command

uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"

will work (if *diff* is a permitted command).

BUGS

Protected files and files that are in protected directories that are owned by the requester can be sent in commands using *uux*. However, if the requester is **root**, and the directory is not searchable by "other", the request will fail.

NAME

uuxqt – execute remote command requests

SYNOPSIS

/usr/lib/uucp/uuxqt [**–s** system] [**–x** debug_level]

DESCRIPTION

The *uuxqt* command executes remote job requests from remote systems generated by the use of the *uux* command. (*Mail* uses *uux* for remote mail requests.) *uuxqt* searches the spool directories looking for *X.* files. For each *X.* file, *uuxqt* checks to see if all the required data files are available and accessible, and file commands are permitted for the requesting system. The Permissions file is used to validate file accessibility and command execution permission.

There are two environment variables that are set before the *uuxqt* command is executed:

UU_MACHINE is the machine that sent the job (the previous one).
UU_USER is the user that sent the job.

These can be used in writing commands that remote systems can execute to provide information, auditing, or restrictions.

The **–x** *debug_level* is a single digit between 0 and 9. Higher numbers give more detailed debugging information.

FILES

/usr/lib/uucp/Permissions
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*

SEE ALSO

mail(1), uucico(1M), uucp(1C), uustat(1C), uux(1C).

NAME

vi, view, vedit – screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [**–t** tag] [**–r** file] [**–L**] [**–wn**] [**–R**] [**–x**] [**–C**] [**–c** command] file ...

view [**–t** tag] [**–r** file] [**–L**] [**–wn**] [**–R**] [**–x**] [**–C**] [**–c** command] file ...

vedit [**–t** tag] [**–r** file] [**–L**] [**–wn**] [**–R**] [**–x**] [**–C**] [**–c** command] file ...

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. The visual commands are described on this manual page; how to set options (like automatically numbering lines and automatically starting a new output line when you type carriage return) and all *ex*(1) line editor commands are described on the *ex*(1) manual page.

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.

Invocation Options

The following invocation options are interpreted by *vi* (previously documented options are discussed in the **NOTES** section at the end of this manual page):

–t *tag* Edit the file containing the *tag* and position the editor at its definition.

–r *file* Edit *file* after an editor or system crash. (Recovers the version of *file* that was in the buffer when the crash occurred.)

–L List the name of all files saved as the result of an editor or system crash.

–w*n* Set the default window size to *n*. This is useful when using the editor over a slow-speed line.

–R **Read-only** mode; the **read-only** flag is set, preventing accidental overwriting of the file.

–x Encryption option; when used, *vi* simulates the **X** command of *ex*(1) and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of *crypt*(1). The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **–x** option. [See *crypt*(1)]. Also, see the **WARNING** section at the end of this manual page.

–C Encryption option; same as the **–x** option, except that *vi* simulates the **C** command of *ex*(1). The **C** command is like the **X** command of *ex*(1), except that all text read in is assumed to have been encrypted.

−c *command* Begin editing by executing the specified editor *command* (usually a search or positioning command).

The *file* argument indicates one or more files to be edited.

The *view* invocation is the same as *vi* except that the **read-only** flag is set.

The *vedit* invocation is intended for beginners. It is the same as *vi* except that the **report** flag is set to 1, the **showmode** and **novice** flags are set, and **magic** is turned off. These defaults make it easier to learn how to use *vi*.

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 setting any of the following options: **a A i 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 an interrupt.
Last line	Reading input for **: / ?** or **!**; terminate by typing a carriage return; an interrupt cancels termination.

COMMAND SUMMARY

In the descriptions, **CR** stands for carriage return and **ESC** stands for the escape key.

Sample commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i*text***ESC**	insert *text*
cw*new***ESC**	change word to *new*
ea*s***ESC**	pluralize word (end of word; append s; escape from input state)
x	delete a character
dw	delete a word
dd	delete a line
3dd	delete 3 lines
u	undo previous change
ZZ	exit *vi*, saving changes
:q!CR	quit, discarding changes
/*text***CR**	search for *text*
^U ^D	scroll up or down
:*cmd***CR**	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

DEL	(delete or rubout) interrupts

File manipulation

ZZ	if file modified, write and exit; otherwise, exit
:wCR	write back changes
:w!CR	forced write, if permission originally not valid
:qCR	quit
:q!CR	quit, discard changes
:e *name***CR**	edit file *name*
:e!CR	reedit, discard changes
:e + *name***CR**	edit, starting at end
:e +*n***CR**	edit starting at line *n*
:e #CR	edit alternate file
:e! #CR	edit alternate file, discard changes
:w *name***CR**	write file *name*
:w! *name***CR**	overwrite file *name*
:shCR	run shell, then return
:!*cmd***CR**	run *cmd*, then return
:nCR	edit next file in arglist
:n *args***CR**	specify new arglist
^G	show current file and line
:ta *tag***CR**	position cursor to *tag*

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a carriage return.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
*n***G**	go to the beginning of the specified line (end default), where *n* is a line number
/pat	next line matching *pat*
?pat	previous line matching *pat*
n	repeat last / or ? command
N	reverse last / or ? command
/pat/+n	n-th line after *pat*
?pat?−n	n-th line before *pat*
]]	next section/function
[[	previous section/function
(	beginning of sentence
)	end of sentence
{	beginning of paragraph
}	end of paragraph
%	find matching ( ) { or }

Adjusting the screen

^L	clear and redraw window
^R	clear and redraw window if ^L is → key
zCR	redraw screen with current line at top of window
z−CR	redraw screen with current line at bottom of window
z.CR	redraw screen with current line at center of window
*/pat/***z−CR**	move *pat* line to bottom of window

z*n***.CR**	use *n*-line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

``	move cursor to previous context
''	move cursor to first non-white space in line
m*x*	mark current position with the ASCII lower-case letter *x*
\`*x*	move cursor to mark *x*
'*x*	move cursor to first non-white space in line marked by *x*

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
–	previous line, at first non-white
CR	return, same as **+**
↓ or **j**	next line, same column
↑ or **k**	previous line, same column

Character positioning

^	first non white-space character
0	beginning of line
$	end of line
h or **→**	forward
l or **←**	backward
^H	same as **←** (backspace)
space	same as **→** (space bar)
f*x*	find next *x*
F*x*	find previous **x**
t*x*	move to character prior to next *x*
T*x*	move to character following previous *x*
;	repeat last **f F t** or **T**
,	repeat inverse of last **f F t** or **T**
n|	move to column *n*
%	find matching ({) or }

Words, sentences, paragraphs

w	forward a word
b	back a word
e	end of word
)	to next sentence
}	to next paragraph
(	back a sentence
{	back a paragraph
W	forward a blank-delimited word
B	back a blank-delimited word
E	end of a blank-delimited word

Corrections during insert

^H	erase last character (backspace)
^W	erase last word
erase	your erase character, same as **^H** (backspace)
kill	your kill character, erase this line of input
\	quotes your erase and kill characters
ESC	ends insertion, back to command mode
DEL	interrupt, terminates insert mode
^D	backtab one character; reset left margin of *autoindent*
^^D	caret (^) followed by control-d (^D); backtab to beginning of line; do not reset left margin of *autoindent*
0^D	backtab to beginning of line; reset left margin of *autoindent*
^V	quote non-printable character

Insert and replace

a	append after cursor
A	append at end of line
i	insert before cursor
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R*text***ESC**	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.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
!	filter through command

Miscellaneous Operations

C	change rest of line (**c$**)
D	delete rest of line (**d$**)
s	substitute chars (**cl**)
S	substitute lines (**cc**)
J	join lines
x	delete characters (**dl**)
X	delete characters before cursor (**dh**)
Y	yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked; however, if a buffer is named (using the ASCII lower-case letters **a** - **z**), the text in that buffer is put instead.

3yy	yank 3 lines
3yl	yank 3 characters
p	put back text after cursor
P	put back text before cursor
"*x*p	put from buffer *x*
"*x*y	yank to buffer *x*
"*x*d	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d*p	retrieve *d*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.

FILES

/tmp	default directory where temporary work files are placed; it can be changed using the **directory** option (see the *ex(1)* **set** command)
/usr/lib/terminfo/?/*	compiled terminal description data base
/usr/lib/.COREterm/?/*	subset of compiled terminal description data base

NOTES

Two options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard [see *intro*(1)]. A **−r** option that is not followed with an option-argument has been replaced by **−L** and +*command* has been replaced by **−c** *command*.

SEE ALSO

ed(1), edit(1), ex(1).

User's Guide.

curses/terminfo chapter of the *Programmer's Guide*.

WARNINGS

The encryption options are provided with the Security Administration Utilities package, which is available only in the United States.

Tampering with entries in **/usr/lib/.COREterm/?/*** or **/usr/lib/terminfo/?/*** (for example, changing or removing an entry) can affect programs such as *vi*(1) that expect the entry to be present and correct. In particular, removing the "dumb" terminal may cause unexpected problems.

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.

NAME

volcopy — make literal copy of file system

SYNOPSIS

/etc/volcopy [options] fsname srcdevice volname1 destdevice volname2

DESCRIPTION

The *volcopy* command 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.

The program requests length and density information if it is not given on the command line or is not recorded on an input tape label. If the file system is too large to fit on one reel, *volcopy* will prompt for additional reels. Labels of all reels are checked. Tapes may be mounted alternately on two or more drives. 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 filsystem being copied.

The *srcdevice* or *destdevice* should be the physical disk section or tape (e.g.: **/dev/dsk/0s1** etc.).

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.

Srcdevice and *volname1* are the device and volume from which the copy of the file system is being extracted. *Destdevice* 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];**).

FILES

/etc/log/filesave.log a record of file systems/volumes copied

SEE ALSO

labelit(1M), sh(1).
fs(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

wait – await completion of process

SYNOPSIS

wait [n]

DESCRIPTION

Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

The shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1).

CAVEAT

If you get the error message *cannot fork, too many processes*, try using the *wait*(1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login and the number the system can keep track of.)

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

If *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

wall − write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

The *wall* command 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 *mesg*(1)].

FILES

/dev/tty*

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

"Cannot send to ..." when the open on a user's tty file fails.

NAME

wc – word count

SYNOPSIS

wc [–lwc] [names]

DESCRIPTION

The *wc* command 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

who − who is on the system

SYNOPSIS

who [−uTlHqpdbrtas] [file]

who am i

who am I

DESCRIPTION

The *who* command 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 at login time to obtain its information. If *file* is given, that file [which must be in *utmp*(4) format] is examined. Usually, *file* will be **/etc/wtmp**, which contains a history of all the logins since the file was last created.

The *who* command with the **am i** or **am I** option identifies the invoking user.

The general format for output is:

name [state] line time [idle] [pid] [comment] [exit]

The *name*, *line*, and *time* information is produced by all options except **−q**; the *state* information is produced only by **−T**; the *idle* and *pid* information is produced only by **−u** and **−l**; and the *comment* and *exit* information is produced only by **−a**. The information produced for **−p**, **−d**, and **−r** is explained during the discussion of each option below.

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 *idle* column contains 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 **−s** 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 login. 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 *idle* 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 respawned 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. In addition, it produces the process termination status, process id, and process exit status [see *utmp*(4)] under the *idle*, *pid*, and *comment* headings, respectively.

−t This option indicates the last change to the system clock [via the *date*(1) command] by **root**. See *su*(1M).

−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.

Note to the super-user: After a shutdown to the single-user state, *who* returns a prompt; the reason is that since **/etc/utmp** is updated at login time and there is no login in single-user state, *who* cannot report accurately on this state. *who am i*, however, returns the correct information.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), init(1M), login(1), mesg(1), su(1M).
wait(2), inittab(4), utmp(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

whodo — who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

The *whodo* command produces formatted and dated output from information in the **/etc/utmp** and **/etc/ps_data** files.

The display is headed by the date, time, and machine name. For each user logged in, device name, user-id, and login time is shown, followed by a list of active processes associated with the user-id. The list includes the device name, process-id, cpu minutes and seconds used, and process name.

EXAMPLE

The command:

```
whodo
```

produces a display like this:

```
Tue Mar 12 15:48:03 1985
bailey

tty09    mcn      8:51
    tty09   28158   0:29 sh

tty52    bdr     15:23
    tty52   21688   0:05 sh
    tty52   22788   0:01 whodo
    tty52   22017   0:03 vi
    tty52   22549   0:01 sh

xt162    lee     10:20
    tty08    6748   0:01 layers
    xt162    6751   0:01 sh
    xt163    6761   0:05 sh
    tty08    6536   0:05 sh
```

FILES

/etc/passwd
/etc/ps_data
/etc/utmp

SEE ALSO

ps(1), who(1).

NAME

write — write to another user

SYNOPSIS

write user [line]

DESCRIPTION

The *write* command 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, such as *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).

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.

NAME

wtinit – object downloader for the 5620 DMD terminal

SYNOPSIS

/usr/lib/layersys/wtinit [–d] [–p] file

DESCRIPTION

The *wtinit* utility downloads the named *file* for execution in the AT&T TELETYPE 5620 DMD terminal connected to its standard output. *file* must be a DMD object file. *wtinit* performs all necessary bootstrap and protocol procedures.

There are two options:

–d Prints out the sizes of the text, data, and bss portions of the downloaded *file* on standard error.

–p Prints the downloading protocol statistics and a trace on standard error.

The environment variable **JPATH** is the analog of the shell's **PATH** variable to define a set of directories in which to search for *file*.

If the environment variable **DMDLOAD** has the value **hex**, *wtinit* will use a hexadecimal download protocol that uses only printable characters.

Terminal Feature Packages for specific versions of AT&T windowing terminals will include terminal-specific versions of *wtinit* under those installation sub-directories. */usr/lib/layersys/wtinit* is used for *layers*(1) initialization only when no Terminal Feature Package is in use.

EXIT STATUS

Returns **0** upon successful completion, **1** otherwise.

WARNING

Standard error should be redirected when using the **–d** or **–p** options.

SEE ALSO

layers(1).

NAME

xargs − construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

The *xargs* command 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 arguments. This process is repeated until there are no more arguments. 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.

−s*size* 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.

−e*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.

The *xargs* command 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 example 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 example 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).

NAME

xfsck – check and repair XENIX filesystems

SYNOPSIS

/bin/xfsck [options] [filesystem] ...

DESCRIPTION

The *xfsck* command audits and interactively repairs inconsistent conditions for XENIX System V filesystems. If the filesystem is consistent, then *xfsck* reports number of files, number of blocks used, and number of blocks free. If the filesystem is inconsistent, the user is prompted whether or not *xfsck* should proceed with each correction. It should be noted that most corrective actions result in some loss of data. The amount and severity of the loss can be determined from the diagnostic output. If the user does not have write permission, *xfsck* defaults to the action of the **–n** option.

The *xfsck* options are:

-y Assumes a response to all questions asked by *xfsck.*

-n Assumes a response to all questions asked by *xfsck.* This option does not open the filesystem for writing.

-s *b*:*c* Ignores the actual free list and unconditionally reconstructs a new one by rewriting the super-block of the filesystem. The filesystem *must* be unmounted while this is done.

This option allows for creating an optimal free-list organization. The following forms are supported:

-s

-sBlocks-per-cylinder:Blocks-to-skip (filesystem interleave)

If *b*:*c* is not given, then the values that were used when the filesystem was created are used again. If these values were not specified, then the default value is used.

-S Conditionally reconstructs the free list. This option is similar to **-s** *b*:*c* above, except that the free list is rebuilt only if there are no discrepancies discovered in the filesystem. The **-S** option forces a "no" response to all questions asked by *xfsck.* This option is useful for forcing free-list reorganization on uncontaminated filesystems.

-t Causes *xfsck* to use the next argument as the scratch file, if needed. A scratch file is used if *xfsck* cannot obtain enough memory to keep its tables. Without the **-t** flag, *xfsck* prompts the user for the name of the scratch file. The file chosen should not be on the filesystem being checked. In addition, if the scratch file is not a special file or did not already exist, it is removed when *xfsck* completes. Note that if the system has a large hard disk, there may not be enough space on another filesystem for the scratch file. In such cases, if the system has a floppy disk drive, use a blank, formatted floppy disk in the floppy disk drive with (for example) **/dev/fd0** specified as the scratch file.

-q Causes *xfsck* to perform a quiet check. Does not print size-check messages in Phase 1. Unreferenced **fifo5** files are selectively removed. If *xfsck* requires it, counts in the superblock are automatically fixed and the free list salvaged.

-D Checks directories for bad blocks. Use this option after the system crashes.

-f Causes *xfsck* to perform a fast check. *xfsck* checks block and sizes (Phase 1) and checks the free list (Phase 5). The free list is reconstructed (Phase 6), if necessary.

-rr Recovers the root filesystem. The required *filesystem* argument must refer to the root filesystem, and preferably to the block device (normally **/dev/root**). This switch implies **-y** (yes) and overrides **-n** (no). If any modifications to the filesystem are required, the system will be automatically shutdown to ensure the integrity of the filesystem.

-c Causes any supported filesystem to be converted to the current filesystem type. The user is prompted to verify the conversion of each filesystem, unless the **-y** option is specified. It is recommended that every filesystem be checked with this option *while unmounted* if it is to be used with the current version of XENIX. To update the active root filesystem, check it with the following command line:

```
xfsck -c —rr /dev/root
```

If no *filesystems* are specified, *xfsck* reads a list of default filesystems from the **/etc/checklist** file.

The following are some of the inconsistencies *xfsck* checks for:

- Blocks claimed by more than one inode or the free list
- Blocks claimed by an inode or the free list outside the range of the filesystem
- Incorrect link counts
- Size checks:
  Incorrect number of blocks
  Directory size not 16-byte aligned
- Bad inode format
- Blocks not accounted for anywhere
- Directory checks:
  File pointing to unallocated inode
  Inode number out of range
- Super block checks:
  More than 65536 inodes
  More blocks for inodes than there are in the filesystem
- Bad free block list format
- Total free block or free inode count incorrect

With the user's consent, *xfsck* reconnects orphaned (allocated, but unreferenced) files and directories by placing them in the **lost+found** directory. The file's (or directory's) inode number then becomes its name. Note that the **lost+found** directory must already exist in the root of the filesystem being checked and must have empty slots in which entries can be made. To create the **lost+found** directory, copy a few files to the directory, then remove them (before executing *xfsck*).

FILES

/etc/checklist	Contains default list of filesystems to check
/etc/default/boot	Contains flags for automatic boot control

SEE ALSO

fsck(1M).

NOTES

xfsck will not run on a mounted non-raw filesystem, unless the filesystem is the root filesystem, or the **-n** option is specified and no writing out of the filesystem will take place. If any such attempt is made, *xfsck* displays a warning and no further processing of the filesystem is done for the specified device.

WARNINGS

xfsck does not support filesystems created under XENIX-86 version 3.0 because the word order in type *long* variables has changed. However, *xfsck* is capable of auditing and repairing XENIX version 3.0 filesystems if the word ordering is correct.

Run **xfsck -rr /dev/root** for the root filesystem. Run **xfsck /dev/??** on the *unmounted* block device for all other filesystems.

It is not recommended that users use *xfsck* on raw devices. Although checking a raw device is almost always faster, there is no way to tell if the filesystem is mounted. If the filesystem is mounted, cleaning it will almost certainly result in an inconsistent superblock.

NAME

xinstall – XENIX installation shell script

SYNOPSIS

/etc/xinstall [device]

DESCRIPTION

/etc/xinstall is the *sh*(1) script used to install XENIX distribution (or application program) floppies. It performs the following tasks:

- Prompts for insertion of floppies.

- Extracts files using the *tar*(1) utility.

- Executes **/once/init.*** programs on each floppy after they have been extracted.

- Removes any **/once/init.*** programs when the installation is finished.

The optional argument to the command specifies the device used. The default device is **/dev/xinstall** and is normally linked to **/dev/rdsk/f0q15dt**.

FILES

/etc/xinstall

/once/init.*

SEE ALSO

custom(1M), fixperm(1M), installpkg(1).

NOTES

xinstall is provided for use with any existing XENIX packages you may have that you wish to install on the UNIX operating system. *xinstall* does not work with UNIX system applications [use *installpkg*(1) to install UNIX system applications].

NAME

xrestore, xrestor – invoke XENIX incremental filesystem restorer

SYNOPSIS

xrestore *key* [*arguments*]

xrestor *key* [*arguments*]

DESCRIPTION

xrestore is used to read archive media backed up with the XENIX *backup*(C) command. The *key* specifies what is to be done. *Key* is one of the characters **rRxt**, optionally combined with **f**. *xrestor* is an alternate spelling for the same command.

f Uses the first *argument* as the name of the archive instead of the default.

F*num* Specifies the file number of the first volume to be restored.

k*vsize* Specifies the size of the volume to be restored.

r,R The archive is read and loaded into the filesystem specified in *argument*. This should not be done lightly (see below). If the *key* is **R**, *xrestore* asks which archive of a multivolume set to start on. This allows *xrestore* to be interrupted and then restarted (an *fsck* must be done before the restart).

x Each file on the archive named by an *argument* is extracted. The filename has all "mount" prefixes removed; for example, if **/usr** is a mounted filesystem, **/usr/bin/lpr** is named **/bin/lpr** on the archive. The extracted file is placed in a file with a numeric name supplied by *xrestore* (actually the inode number). In order to keep the amount of archive read to a minimum, the following procedure is recommended:

1. Mount volume 1 of the set of backup archives.
2. Type the *xrestore* command.
3. *Restore* will announce whether or not it found the files, give the numeric name that it will assign to the file, and in the case of a tape, rewind to the start of the archive.
4. It then asks you to "mount the desired tape volume". Type the number of the volume you choose. On a multivolume backup the recommended procedure is to mount the volumes, last through first. *Restore* checks to see if any of the requested files are on the mounted archive (or a later archive–thus the reverse order). If the requested files are not there, *xrestore* doesn't read through the tape. If you are working with a single-volume backup or if the number of files being restored is large, respond to the query with **1**, and *xrestore* will read the archives in sequential order.

X*files* Puts files in the directory specified by *arguments*.

t Prints the date the archive was written and the date the filesystem was backed up.

T This causes *xrestore* to behave like *dumpdir*(C) except that it doesn't list directories.

The **r** option should only be used to restore a complete backup archive onto a clear filesystem, or to restore an incremental backup archive onto a filesystem so created. Thus:

```
/etc/mkfs /dev/dsk/0s3 10000
xrestore r /dev/dsk/0s3
```

is a typical sequence to restore a complete backup. Another *xrestore* can be done to get an incremental backup in on top of this.

A *backup* followed by a *mkfs* and a *xrestore* is used to change the size of a filesystem.

FILES

rst*	Temporary files
/etc/default/xrestore	Name of default archive device

The default archive unit varies with installation.

NOTES

xrestore is for XENIX compatibility and should only be used to restore filesystems that were backed up under XENIX.

It is not possible to successfully restore an entire active **root** filesystem.

DIAGNOSTICS

There are various diagnostics involved with reading the archive and writing the disk. There are also diagnostics if the i-list or the free list of the filesystem is not large enough to hold the dump.

If the dump extends over more than one disk or tape, it may ask you to change disks or tapes. Reply with a NEWLINE when the next unit has been mounted.

NAME

xtd – extract and print xt driver link structure

SYNOPSIS

xtd [–f] [–n ...]

DESCRIPTION

The *xtd* command is a debugging tool for the *xt*(7) driver. It performs an **XTIOCDATA** *ioctl*(2) call on its standard input file to extract the *Link* data structure for the attached group of channels. This call will fail if data extraction has not been configured in the driver or the standard input is not attached to an *xt*(7) channel. The data are printed one item per line on the standard output. The output should probably be formatted via **pr –3**.

The optional flags affect output as follows:

–n — *n* is a number in the range 0 to 7. Channel *n* is included in the list of channels to be printed. The default prints all channels, whereas the occurrence of one or more channel numbers implies a subset.

–f — Causes a "formfeed" character to be put out at the end of the output for the benefit of page-display programs.

EXIT STATUS

Returns **0** upon successful completion; **1** otherwise.

SEE ALSO

pr(1), xts(1M), xtt(1M), xt(7).

ioctl(2), xtproto(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

xts − extract and print xt driver statistics

SYNOPSIS

xts [−f]

DESCRIPTION

The *xts* command is a debugging tool for the *xt*(7) driver. It performs an **XTIOCSTATS** *ioctl*(2) call on its standard input file to extract the accumulated statistics for the attached group of channels. This call will fail if statistics have not been configured in the driver, or the standard input is not attached to an *xt*(7) channel. The statistics are printed one item per line on the standard output.

−f Causes a "formfeed" character to be put out at the end of the output for the benefit of page-display programs.

EXIT STATUS

Returns **0** upon successful completion; **1** otherwise.

SEE ALSO

xtd(1M), xtt(1M), xt(7).

ioctl(2), xtproto(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

xtt – extract and print xt driver packet traces

SYNOPSIS

xtt [–f] [–o]

DESCRIPTION

The *xtt* command is a debugging tool for the *xt*(7) driver. It performs an **XTIOCTRACE** *ioctl*(2) call on its standard input file to turn on tracing and extract the circular packet trace buffer for the attached group of channels. This call will fail if tracing has not been configured in the driver, or the standard input is not attached to an *xt*(7) channel. The packets are printed on the standard output.

The optional flags are:

–f Causes a "formfeed" character to be put out at the end of the output for the benefit of page-display programs.

–o Turns off further driver tracing.

EXIT STATUS

Returns **0** upon successful completion; **1** otherwise.

NOTE

If driver tracing has not been turned on for the terminal session by invoking *layers*(1) with the **–t** option, *xtt* will not generate any output the first time it is executed.

SEE ALSO

layers(1), xtd(1M), xts(1M), xt(7).

ioctl(2), layers(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

yes — repeatedly print string

SYNOPSIS

yes [*string*]

DESCRIPTION

The *yes* command repeatedly outputs "y", or if a single string argument is given, *arg* is output. The command continues indefinitely unless aborted. *Yes* is useful in pipes to commands that prompt for input and require a "y" response for a yes. In this case, *yes* terminates when the command that it pipes to terminates, so that no infinite loop occurs.

NAME

cftime – language specific strings

DESCRIPTION

The programmer can create one printable file per language. These files must be kept in a special directory /lib/cftime. If this directory does not exist, the programmer should create it. The contents of these files are:

- abbreviated month names (in order)
- month names (in order)
- abbreviated weekday names (in order)
- weekday names (in order)
- default strings that specify formats for local time (%**x**) and local date (%**X**).
- default format for cftime, if the argument for cftime is zero or null.
- AM (ante meridian) string
- PM (post meridian) string

Each string is on a line by itself. All white space is significant. The order of the strings in the above list is the same order in which the strings appear in the file shown below.

EXAMPLE

/lib/cftime/usa_english

```
Jan
Feb
 ...
January
February
 ...
Sun
Mon
 ...
Sunday
Monday
 ...
%H:%M:%S
%m/%d/%y
%a %b %d %T %Z %Y
AM
PM
```

FILES

/lib/cftime – directory that contains the language specific printable files (create it if it does not exist)

SEE ALSO

ctime(3C).

NAME

gettydefs — speed and terminal settings used by getty

DESCRIPTION

The **/etc/gettydefs** file contains information used by *getty*(1M) to set up the speed and terminal settings for a line. It supplies information on what the *login*(1) prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by typing a <*break*> character.

NOTE: Customers who need to support terminals that pass 8 bits to the system (as is typical outside the U.S.A.) must modify the entries in **/etc/gettydefs** as described in the **WARNINGS** section.

Each entry in **/etc/gettydefs** has the following format:

label# initial-flags # final-flags # login-prompt #next-label

Each entry is followed by a blank line. The various fields can contain quoted characters of the form **\b**, **\n**, **\c**, etc., as well as \ *nnn*, where *nnn* is the octal value of the desired character. The various fields are:

label This is the string against which *getty*(1M) tries to match its second argument. It is often the speed, such as **1200**, at which the terminal is supposed to run, but it need not be (see below).

initial-flags These flags are the initial *ioctl*(2) settings to which the terminal is to be set if a terminal type is not specified to *getty*(1M). The flags that *getty*(1M) understands are the same as the ones listed in **/usr/include/sys/termio.h** [see *termio*(7)]. Normally only the speed flag is required in the *initial-flags*. *getty*(1M) automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-flag* settings remain in effect until *getty*(1M) executes *login*(1).

final-flags These flags take the same values as the *initial-flags* and are set just before *getty*(1M) executes *login*(1). The speed flag is again required. The composite flag **SANE** takes care of most of the other flags that need to be set so that the processor and terminal are communicating in a rational fashion. The other two commonly specified *final-flags* are **TAB3**, so that tabs are sent to the terminal as spaces, and **HUPCL**, so that the line is hung up on the final close.

login-prompt This entire field is printed as the *login-prompt*. Unlike the above fields where white space is ignored (a space, tab or new-line), they are included in the *login-prompt* field.

next-label If this entry does not specify the desired speed, indicated by the user typing a <*break*> character, then *getty*(1M) will search for the entry with *next-label* as its *label* field and set up the terminal for those settings. Usually, a series of speeds are linked together in this fashion, into a closed set; for instance, **2400** linked to **1200**, which in turn is linked to **300**, which finally is linked to **2400**.

If *getty*(1M) is called without a second argument, then the first entry of **/etc/gettydefs** is used, thus making the first entry of **/etc/gettydefs** the default entry. It is also used if *getty*(1M) cannot find the specified *label*. If **/etc/gettydefs** itself is missing, there is one entry built into *getty*(1M) which will bring up a terminal at **300** baud.

It is strongly recommended that after making or modifying **/etc/gettydefs**, it be run through *getty*(1M) with the check option to be sure there are no errors.

FILES

/etc/gettydefs

SEE ALSO

ioctl(2).
getty(1M), login(1), stty(1), termio(7) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

WARNINGS

To support terminals that pass 8 bits to the system (also, see the **BUGS** section), modify the entries in the **/etc/gettydefs** file for those terminals as follows: add **CS8** to *initial-flags* and replace all occurrences of **SANE** with the values: **BRKINT IGNPAR ICRNL IXON OPOST ONCLR CS8 ISIG ICANON ECHO ECHOK**

An example of changing an entry in **/etc/gettydefs** is illustrated below. All the information for an entry must be on one line in the file.

Original entry:

CONSOLE # B9600 HUPCL OPOST ONLCR # B9600 SANE IXANY TAB3 HUPCL # Console Login: # console

Modified entry:

CONSOLE # B9600 CS8 HUPCL OPOST ONLCR # B9600 BRKINT IGNPAR ICNRL IXON OPOST ONLCR CS8 ISIG ICANON ECHO ECHOK IXANY TAB3 HUPCL # Console Login: # console

This change will permit terminals to pass 8 bits to the system so long as the system is in MULTI-USER state. When the system changes to SINGLE-USER state, the *getty*(1M) is killed and the terminal attributes are lost. So to permit a terminal to pass 8 bits to the system in SINGLE-USER state, after you are in SINGLE-USER state, type [see *stty*(1)]:

stty –istrip cs8

BUGS

8-bit with parity mode is not supported.

NAME

group — group file

DESCRIPTION

group contains for each group the following information:

group name
encrypted password
numerical group ID
comma-separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a new-line. If the password field is null, no password is demanded.

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 group ID's to names.

FILES

/etc/group

SEE ALSO

passwd(4).
newgrp(1M), passwd(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

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 *getty*s 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 up to four 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 the first time *init* goes from single-user to multi-user state after the system is booted. (If **initdefault** is set to **2**, the process will run right after the boot.) *Init* starts the process, waits for its termination and, when it dies, does 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**. 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 (i.e., before the **Console Login:** prompt). It is expected that this entry will be used only 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).

getty(1M), init(1M), sh(1), sulogin(1M), who(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

NAME

issue – issue identification file

DESCRIPTION

The file **/etc/issue** contains the *issue* or project identification to be printed as a login prompt. This is an ASCII file which is read by program *getty* and then written to any terminal spawned or respawned from the *lines* file.

FILES

/etc/issue

SEE ALSO

login(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

NAME

/usr/adm/loginlog — log of failed login attempts

DESCRIPTION

After five unsuccessful login attempts, all the attempts are logged in the **loginlog** file. This file contains one record for each failed attempt. Each record contains the following information:

login name
tty specification
time

This is an ASCII file. Each field within each entry is separated from the next by a colon. Each entry is separated from the next by a newline.

By default, **loginlog** does not exist, so no logging is done. To enable logging, the log file must be created with read and write permission for owner only. Owner must be **root** and group must be **sys**.

FILES

/usr/adm/loginlog

SEE ALSO

login(1), passwd(1), passwd(1M) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

NAME

mdevice − file format

SYNOPSIS

mdevice

DESCRIPTION

The *mdevice* file is included in the directory */etc/conf/cf.d*. It includes a one-line description of each device driver and configurable software module in the system to be built [except for file system types, see *mfsys*(4)]. Each line in *mdevice* represents the *Master* file component from a Driver Software Package (DSP) either delivered with the base system or installed later via *idinstall*.

Each line contains several whitespace-separated fields; they are described below. Each field must be supplied with a value or a '−' (dash).

1. *Device name*: This field is the internal name of the device or module, and may be up to 8 characters long. The first character of the name must be an alphabetic character; the others may be letters, digits, or underscores.

2. *Function list*: This field is a string of characters that identify driver functions that are present. Using one of the characters below requires the driver to have an entry point (function) of the type indicated. If no functions in the following list are supplied, the field should contain a dash.

   o − open routine
   c − close routine
   r − read routine
   w − write routine
   i − ioctl routine
   s − startup routine
   x − exit routine
   f − fork routine
   e − exec routine
   I − init routine
   h − halt routine
   p − poll routine
   E − enter routine
   X − exit routine

   Note that if the device is a 'block' type device (see field 3. below), a *strategy* routine and a *print* routine are required by default.

3. *Characteristics of driver*: This field contains a set of characters that indicate the characteristics of the driver. If none of the characters below apply, the field should contain a dash. The legal characters for this field are:

- i — The device driver is installable.
- c — The device is a 'character' device.
- b — The device is a 'block' device.
- t — The device is a tty.
- o — This device may have only one *sdevice* entry.
- r — This device is required in all configurations of the Kernel. This option is intended for drivers delivered with the base system only. Device nodes (special files in the */dev* directory), once made for this device, are never removed. See *idmknod.*
- S — This device driver is a STREAMS module.
- H — This device driver controls hardware. This option distinguishes drivers that support hardware from those that are entirely software (pseudo-devices).
- G — This device does not use an interrupt though an interrupt is specified in the *sdevice* entry. This is used when you wish to associate a device to a specific device group.
- D — This option indicates that the device driver can share its DMA channel.
- O — This option indicates that the IOA range of this device may overlap that of another device.

4. *Handler prefix*: This field contains the character string prepended to all the externally-known handler routines associated with this driver. The string may be up to 4 characters long.
5. *Block Major number*: This field should be set to zero in a DSP *Master* file. If the device is a 'block' type device, a value will be assigned by *idinstall* during installation.
6. *Character Major number*: This field should be set to zero in a DSP *Master* file. If the device is a 'character' type device (or 'STREAMS' type), a value will be assigned by *idinstall* during installation.
7. *Minimum units*: This field is an integer specifying the minimum number of these devices that can be specified in the *sdevice* file.
8. *Maximum units*: This field specifies the maximum number of these devices that may be specified in the *sdevice* file. It contains an integer.
9. *DMA channel*: This field contains an integer that specifies the DMA channel to be used by this device. If the device does not use DMA, place a '−1' in this field. Note that more than one device can share a DMA channel (previously disallowed).

SPECIFYING STREAMS DEVICES AND MODULES

STREAMS modules and drivers are treated in a slightly different way from other drivers in all UNIX systems, and their configuration reflects this difference. To specify a STREAMS device driver, its *mdevice* entry should contain both an 'S' and a 'c' in the *characteristics* field (see 3. above). This indicates that it is a STREAMS driver and that it requires an entry in the UNIX kernel's *cdevsw* table, where STREAMS drivers are normally configured into the system.

A STREAMS module that is not a device driver, such as a line discipline module, requires an 'S' in the *characteristics* field of its *mdevice* file entry, but should not include a 'c', as a device driver does.

SEE ALSO

mfsys(4), sdevice(4).

idinstall(1M) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

NAME

mfsys — file format

SYNOPSIS

mfsys

DESCRIPTION

The *mfsys* file contains configuration information for file system types that are to be included in the next system kernel to be built. It is included in the directory */etc/conf/cf.d*, and includes a one-line description of each file system type. The *mfsys* file is coalesced from component files in the directory */etc/conf/mfsys.d*. Each line contains the following whitespace-separated fields:

1. *name*: This field contains the internal name for the file system type (e.g., S51K, DUFST). This name is no more than 32 characters long, and by convention is composed of uppercase alphanumeric characters.
2. *prefix*: The *prefix* in this field is the string prepended to the *fstypsw* handler functions defined for this file system type (e.g., s5, du). The prefix must be no more that 8 characters long.
3. *flags*: The *flags* field contains a hex number of the form "0xNN" to be used in populating the *fsinfo* data structure table entry for this file system type.
4. *notify flags*: The *notify flags* field contains a hex number of the form "0xNN" to be used in population the *fsinfo* data structure table entry for this file system type.
5. *function bitstring*: The *function bitstring* is a string of 28 0's and 1's. Each file system type potentially defines 28 functions to populate the *fstypsw* data structure table entry for itself. All file system types do not supply all the functions in this table, however, and this bitstring is used to indicate which of the functions are present and which are absent. A '1' in this string indicates that a function has been supplied, and a '0' indicates that a function has not been supplied. Successive characters in the string represent successive elements of the *fstypsw* data structure, with the first entry in this data structure represented by the rightmost character in the string.

SEE ALSO

sfsys(4).
idinstall(1M), idbuild(1M) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

NAME

mtune – file format

SYNOPSIS

mtune

DESCRIPTION

The *mtune* file contains information about all the system tunable parameters. Each tunable parameter is specified by a single line in the file, and each line contains the following whitespace-separated set of fields:

1. *parameter name*: A character string no more than 20 characters long. It is used to construct the preprocessor "#define's" that pass the value to the system when it is built.
2. *default value*: This is the default value of the tunable parameter. If the value is not specified in the *stune* file, this value will be used when the system is built.
3. *minimum value*: This is the minimum allowable value for the tunable parameter. If the parameter is set in the *stune* file, the configuration tools will verify that the new value is equal to or greater than this value.
4. *maximum value*: This is the maximum allowable value for the tunable parameter. If the parameter is set in the *stune* file, the configuration tools will check that the new value is equal to or less than this value.

The file *mtune* normally resides in */etc/conf/cf.d*. However, a user or an add-on package should never directly edit the *mtune* file to change the setting of a system tunable parameter. Instead the *idtune* command should be used to modify or append the tunable parameter to the *stune* file.

In order for the new values to become effective, the UNIX system kernel must be rebuilt and the system must then be rebooted.

SEE ALSO

stune(4).

idbuild(1M), idtune(1M) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.

NAME

passwd — password file

DESCRIPTION

passwd contains for each user the following information:

login name
password and (optional) aging
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 shell field is null, **/bin/sh** is used.

This file has user login information, and has general read permission. It can therefore be used, for example, to map numerical user IDs to names.

The password field consists of the character **x** if there is a **/etc/shadow** file. If **/etc/shadow** does not exist and the login does have a password, this field will contain an encrypted copy of the password. This field remains only for compatibility reasons when **/etc/shadow** exists.

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 his encrypted password in the password 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. A user who attempts to login after his password has expired will be forced to supply a new one. The next character, *m* say, denotes the minimum period in weeks that 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 ..) the user will be forced to change his password the next time he logs in (and the "age" will disappear from his entry in the password file). If *m* > *M* (signified, for example, by the string ./) only the super-user will be able to change the password.

FILES

/etc/passwd
/etc/shadow

SEE ALSO

getpwent(3C), group(4).

login(1), passwd(1), and passwd(1M) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

NAME

profile – setting up an environment at login time

SYNOPSIS

/etc/profile
$HOME/.profile

DESCRIPTION

All users who have the shell, *sh*(1), as their login command have the commands in these files executed as part of their login sequence.

/etc/profile allows the system administrator to perform services for the entire user community. Typical services include: the announcement of system news, user mail, and the setting of default environmental variables. It is not unusual for */etc/profile* to execute special actions for the **root** login or the *su*(1M) command. Computers running outside the Eastern time zone should have the line

```
. /etc/TIMEZONE
```

included early in /etc/profile [see timezone(4)].

The file *$HOME/.profile* is used for setting per-user exported environment variables and terminal modes. The following example is typical (except for the comments):

```
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 027
# Tell me when new mail comes in
MAIL=/usr/mail/$LOGNAME
# Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Set terminal type
while :
do  echo "terminal: \c"
    read TERM
    if [ -f ${TERMINFO:-/usr/lib/terminfo}/?/$TERM ]
    then break
    elif [ -f /usr/lib/terminfo/?/$TERM ]
    then break
    else echo "invalid term $TERM" 1>&2
    fi
done
# Initialize the terminal and set tabs
# The environmental variable TERM must have been exported
# before the "tput init" command is executed.
tput init
# Set the erase character to backspace
stty erase '^H' echoe
```

FILES

/etc/TIMEZONE	timezone environment
$HOME/.profile	user-specific environment
/etc/profile	system-wide environment

SEE ALSO

terminfo(4), timezone(4), environ(5), term(5).

env(1), login(1), mail(1), sh(1), stty(1), su(1M), tput(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

User's Guide.

Programmer's Guide.

NOTES

Care must be taken in providing system-wide services in */etc/profile.* Personal *.profile* files are better for serving all but the most global needs.

NAME

sdevice – file format

SYNOPSIS

sdevice

DESCRIPTION

The *sdevice* file contains local system configuration information for each of the devices specified in the *mdevice* file. It contains one or more entries for each device specified in *mdevice*. *Sdevice* is present in the directory */etc/conf/cf.d*, and is coalesced from component files in the directory */etc/conf/sdevice.d*. Files in */etc/conf/sdevice.d* are the *System* file components either delivered with the base system or installed later via *idinstall*.

Each entry must contain the following whitespace-separated fields:

1. *Device name:* This field contains the internal name of the driver. This must match one of the names in the first field of an *mdevice* file entry.
2. *Configure:* This field must contain the character 'Y' indicating that the device is to be installed in the kernel. For testing purposes, an 'N' may be entered indicating that the device will not be installed.
3. *Unit:* This field can be encoded with a device dependent numeric value. It is usually used to represent the number of subdevices on a controller or pseudo-device. Its value must be within the minimum and maximum values specified in fields 7 and 8 of the *mdevice* entry.
4. *Ipl*: The *ipl* field specifies the system ipl level at which the driver's interrupt handler will run in the new system kernel. Legal values are 0 through 8. If the driver doesn't have an interrupt handling routine, put a 0 in this field.
5. *Type:* This field indicates the type of interrupt scheme required by the device. The permissible values are:

   0 – The device does not require an interrupt line.

   1 – The device requires an interrupt line.
   If the driver supports more than one hardware controller, each controller requires a separate interrupt.

   2 – The device requires an interrupt line.
   If the driver supports more than one hardware controller, each controller will share the same interrupt.

   3 – The device requires an interrupt line.
   If the driver supports more than one hardware controller, each controller will share the same interrupt. Multiple device drivers having the same ipl level can share this interrupt.
6. *Vector:* This field contains the interrupt vector number used by the device. If the *Type* field contains a 0 (i.e., no interrupt required), this field should be encoded with a 0. Note that more than one device can share an interrupt number.

7. *SIOA:* The *SIOA* field (Start I/O Address) contains the starting address on the I/O bus through which the device communicates. This field must be within 0x1 and 0x3fff. (If this field is not used, it should be encoded with the value zero.)

8. *EIOA:* The field (End I/O Address) contains the end address on the I/O bus through which the device communicates. This field must be within 0x1 and 0x3fff. (If this field is not used, it should be encoded with the value zero.)

9. *SCMA:* The *SCMA* field (Start Controller Memory Address) is used by controllers that have internal memory. It specifies the starting address of this memory. This field must be within 0xa0000 and 0xfbfff. (If this field is not used, it should be encoded with the value zero.)

10. *ECMA:* The *ECMA* (End Controller Memory Address) specifies the end of the internal memory for the device. This field must be within 0xa0000 and 0xfbfff. (If this field is not used, it should be encoded with the value zero.)

SEE ALSO

mdevice(4).

idinstall(1M) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

NAME

sfsys – file format

SYNOPSIS

sfsys

DESCRIPTION

The *sfsys* file contains local system information about each file system type specified in the *mfsys* file. It is present in the directory */etc/conf/cf.d*, and contains a one-line entry for each file system type specified in the *mfsys* file. The *sfsys* file is coalesced from component files in the directory */etc/conf/sfsys.d.* Each line in this file is a whitespace-separate set of fields that specify:

1. *name*: This field contains the internal name of the file system type (e.g., DUFST, S51K). By convention, this name is up to 32 characters long, and is composed of all uppercase alphanumeric characters.
2. *Y/N*: This field contains either an uppercase 'Y' (for "yes") or an uppercase 'N' (for "no) to indicate whether the named file system type is to be configured into the next system kernel to be built.

SEE ALSO

mfsys(4).
idbuild(1M), idinstall(1M) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

NAME

stune – file format

SYNOPSIS

stune

DESCRIPTION

The *stune* file contains local system settings for tunable parameters. The parameter settings in this file replace the default values specified in the *mtune* file, if the new values are within the legal range for the parameter specified in *mtune*. The file contains one line for each parameter to be reset. Each line contains two whitespace-separated fields:

1. *external name*: This is the external name of the tunable parameter used in the *mtune* file.
2. *value*: This field contains the new value for the tunable parameter.

The file *stune* normally resides in */etc/conf/cf.d*. However, a user or an add-on package should never directly edit the *mtune* file. Instead the *idtune* command should be used.

In order for the new values to become effective the UNIX kernel must be rebuilt and the system must then be rebooted.

SEE ALSO

mtune(4).
idbuild(1M), idtune(1M) in the *INTERACTIVE User's/System Administrator's Reference Manual.*

NAME

timezone – set default system time zone

SYNOPSIS

/etc/TIMEZONE

DESCRIPTION

This file sets and exports the time zone environmental variable **TZ**.

This file is "dotted" into other files that must know the time zone.

The syntax of **TZ** can be described as follows:

```
TZ             →    zone
                  | zone signed_time
                  | zone signed_time zone
                  | zone signed_time zone dst
zone           →    letter letter letter
signed_time    →    sign time
                  | time
time           →    hour
                  | hour : minute
                  | hour : minute : second
dst            →    signed_time
                  | signed_time ; dst_date , dst_date
                  | ; dst_date , dst_date
dst_date       →    julian
                  | julian / time
letter         →    a | A | b | B | ... | z | Z
hour           →    00 | 01 | ... | 23
minute         →    00 | 01 | ... | 59
second         →    00 | 01 | ... | 59
julian         →    001 | 002 | ...| 366
sign           →    – | +
```

EXAMPLES

The contents of **/etc/TIMEZONE** corresponding to the simple example below could be

```
#         Time Zone
TZ=EST5EDT
export TZ
```

A simple setting for New Jersey could be

TZ=EST5EDT

where **EST** is the abbreviation for the main time zone, **5** is the difference, in hours, between GMT (Greenwich Mean Time) and the main time zone, and **EDT** is the abbreviation for the alternate time zone.

The most complex representation of the same setting, for the year 1986, is

TZ="EST5:00:00EDT4:00:00;117/2:00:00,299/2:00:00"

where **EST** is the abbreviation for the main time zone, **5:00:00** is the difference, in hours, minutes, and seconds between GMT and the main time zone, **EDT** is the abbreviation for the alternate time zone, **4:00:00** is the difference, in hours, minutes, and seconds between GMT and the alternate time zone, **117** is the number of the day of the year (Julian day) when the alternate time zone will take effect, **2:00:00** is the number of hours, minutes, and seconds past midnight when the alternate time zone will take effect, **299** is the number of the day of the year when the alternate time zone will end, and **2:00:00** is the number of hours, minutes, and seconds past midnight when the alternate time zone will end.

A southern hemisphere setting such as the Cook Islands could be

TZ="KDT9:30KST10:00;64/5:00,303/20:00"

This setting means that **KDT** is the abbreviation for the main time zone, **KST** is the abbreviation for the alternate time zone, KST is **9** hours and **30** minutes later than GMT, KDT is **10** hours later than GMT, the starting date of KDT is the **64**th day at **5** AM, and the ending date of KDT is the **303**rd day at **8** PM.

Starting and ending times are relative to the alternate time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be midnight.

Note that in most installations, **TZ** is set to the correct value by default when the user logs on, via the local */etc/profile* file [see *profile*(4)].

SEE ALSO

ctime(3C), profile(4), environ(5).
rc2(1M) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

NOTES

When the longer format is used, the **TZ** variable must be surrounded by double quotes as shown.

The system administrator must change the Julian start and end days annually if the longer form of the **TZ** variable is used.

Setting the time during the interval of change from the main time zone to the alternate time zone or vice versa can produce unpredictable results.

NAME

aliases – aliases file for sendmail

SYNOPSIS

/usr/lib/aliases

DESCRIPTION

This file describes user ID aliases used by **/usr/lib/sendmail**. It is formatted as a series of lines of the form

name: name_1, name2, name_3, . . .

The *name* is the name to alias, and the *name_n* are the aliases for that name. Lines beginning with white space are continuation lines. Lines beginning with # are comments.

Aliasing occurs only on local names. Loops cannot occur, since no message will be sent to any person more than once.

After aliasing has been done, local and valid recipients who have a **.forward** file in their home directory have messages forwarded to the list of users defined in that file.

This is only the raw data file; the actual aliasing information is placed into a binary format in the files **/usr/lib/aliases.dir** and **/usr/lib/aliases.pag** using the program *newaliases*(1). A *newaliases* command should be executed each time the aliases file is changed in order for the change to take effect.

SEE ALSO

newaliases(1), sendmail(8) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.
"Sendmail Installation and Operation Guide" and "Sendmail – An Internetwork Mail Router" in the *INTERACTIVE UNIX Operating System Guide*.

BUGS

Because of restrictions in the *dbm* program, a single alias cannot contain more than about 1000 bytes of information. You can get longer aliases by "chaining"; that is, make the last name in the alias be a dummy name which is a continuation alias.

NAME

ascii – map of ASCII character set

DESCRIPTION

ascii is a map of the ASCII character set. It lists octal, hexadecimal, and decimal equivalents of each character. The *ascii* map contains the following tables:

Octal ASCII

000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dc1	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 $	045 %	046 &	047 '
050 (	051)	052 *	053 +	054 ,	055 –	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [	134 \	135]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174 |	175 }	176 ~	177 del

Hexadecimal ASCII

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d –	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

Decimal ASCII

<table>
<tr><td>0 nul</td><td>1 soh</td><td>2 stx</td><td>3 etx</td><td>4 eot</td><td>5 enq</td><td>6 ack</td><td>7 bel</td></tr>
<tr><td>8 bs</td><td>9 ht</td><td>10 nl</td><td>11 vt</td><td>12 np</td><td>13 cr</td><td>14 so</td><td>15 si</td></tr>
<tr><td>16 dle</td><td>17 dc1</td><td>18 dc2</td><td>19 dc3</td><td>20 dc4</td><td>21 nak</td><td>22 syn</td><td>23 etb</td></tr>
<tr><td>24 can</td><td>25 em</td><td>26 sub</td><td>27 esc</td><td>28 fs</td><td>29 gs</td><td>30 rs</td><td>31 us</td></tr>
<tr><td>32 sp</td><td>33 !</td><td>34 "</td><td>35 #</td><td>36 $</td><td>37 %</td><td>38 &</td><td>39 '</td></tr>
<tr><td>40 (</td><td>41)</td><td>42 *</td><td>43 +</td><td>44 ,</td><td>45 -</td><td>46 .</td><td>47 /</td></tr>
<tr><td>48 0</td><td>49 1</td><td>50 2</td><td>51 3</td><td>52 4</td><td>53 5</td><td>54 6</td><td>55 7</td></tr>
<tr><td>56 8</td><td>57 9</td><td>58 :</td><td>59 ;</td><td>60 <</td><td>61 =</td><td>62 ></td><td>63 ?</td></tr>
<tr><td>64 @</td><td>65 A</td><td>66 B</td><td>67 C</td><td>68 D</td><td>69 E</td><td>70 F</td><td>71 G</td></tr>
<tr><td>72 H</td><td>73 I</td><td>74 J</td><td>75 K</td><td>76 L</td><td>77 M</td><td>78 N</td><td>79 O</td></tr>
<tr><td>80 P</td><td>81 Q</td><td>82 R</td><td>83 S</td><td>84 T</td><td>85 U</td><td>86 V</td><td>87 W</td></tr>
<tr><td>88 X</td><td>89 Y</td><td>90 Z</td><td>91 [</td><td>92 \</td><td>93]</td><td>94 ^</td><td>95 _</td></tr>
<tr><td>96 `</td><td>97 a</td><td>98 b</td><td>99 c</td><td>100 d</td><td>101 e</td><td>102 f</td><td>103 g</td></tr>
<tr><td>104 h</td><td>105 i</td><td>106 j</td><td>107 k</td><td>108 l</td><td>109 m</td><td>110 n</td><td>111 o</td></tr>
<tr><td>112 p</td><td>113 q</td><td>114 r</td><td>115 s</td><td>116 t</td><td>117 u</td><td>118 v</td><td>119 w</td></tr>
<tr><td>120 x</td><td>121 y</td><td>122 z</td><td>123 {</td><td>124 |</td><td>125 }</td><td>126 ~</td><td>127 del</td></tr>
</table>

FILES

/usr/pub/ascii

NAME

environ − user environment

DESCRIPTION

An array of strings called the "environment" is made available by *exec*(2) when a process begins. By convention, these strings have the form "name=value". The following names are used by various commands:

CFTIME The default format string to be used by the *date*(1) command and the **ascftime**( ) and **cftime**( ) routines [see *ctime*(3C)]. If **CFTIME** is not set or is null, the default format string specified in the **/lib/cftime/***LANGUAGE* file (if it exists) is used in its place [see *cftime*(4)].

CHRCLASS A value that corresponds to a file in **/lib/chrclass** containing character classification and conversion information. This information is used by commands (such as *cat*(1), *ed*(1), *sort*(1), etc.) to classify characters as alphabetic, printable, uppercase, etc., and to convert characters to uppercase or lowercase.

When a program or command begins execution, the tables containing this information are initialized based on the value of **CHRCLASS**. If **CHRCLASS** is nonexistent, null, set to a value for which no file exists in **/lib/chrclass**, or errors occur while reading the file, the ASCII character set is used. During execution, a program or command can change the values in these tables by calling the **setchrclass**() routine. For more detail, see *ctype*(3C).

These tables are created using the *chrtbl*(1M) command.

HOME The name of the user's login directory, set by *login*(1) from the password file [see *passwd*(4)].

LANGUAGE A language for which a printable file by that name exists in **/lib/cftime.** This information is used by commands (such as *date*(1), *ls*(1), *sort*(1), etc.) to print date and time information in the language specified.

If **LANGUAGE** is nonexistent, null, set to a value for which no file exists in **/lib/cftime**, or errors occur while reading the file, the last language requested will be used. (If no language has been requested, the language **usa_english** is assumed.) For a description of the content of files in **/lib/cftime**, see *cftime*(4).

PATH The sequence of directory prefixes that *sh*(1), *time*(1), *nice*(1), *nohup*(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (:). *login*(1) sets **PATH=:/bin:/usr/bin**. (For more detail, see the "Execution" section of the *sh*(1) manual page.)

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as *mm*(1) or *vi*(1), which may exploit special capabilities of that terminal.

TZ Time zone information. The simplest format is **xxx*n*zzz** where **xxx** is the standard local time zone abbreviation, *n* is the difference in hours from GMT (Greenwich Mean Time), and **zzz** is the abbreviation for an alternate time zone (usually the daylight-saving local time zone), if any; for example,

TZ="EST5EDT"

The most complex format allows you to specify the difference in hours of the alternate time zone from GMT and the starting day and time and ending day and time for using this alternate time zone. For example, in 1985 the complex format corresponding to the above simple example is:

TZ="EST5:00:00EDT4:00:00;118/2:00:00,300/2:00:00"

When the above complex format is used, it must be surrounded by double quotes. For more details, see *ctime*(3C) and *timezone*(4).

Further names may be placed in the environment by the *export* command and "name=value" arguments in *sh*(1), or by *exec*(2). It is unwise to conflict with certain shell variables that are frequently exported by **.profile** files: **MAIL, PS1, PS2, IFS** [see *profile*(4)].

SEE ALSO

exec(2), ctime(3C), ctype(3C), cftime(4), passwd(4), profile(4), timezone(4).

cat(1), date(1), ed(1), env(1), ls(1), login(1), nice(1), nohup(1), sh(1), sort(1), time(1), vi(1), chrtbl(1M) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

mm(1) in the *DOCUMENTER'S WORKBENCH Software Release 2.0 Technical Discussion and Reference Manual.*

NOTES

References to the *cftime*(4), *ctime*(3C), and *ctype*(3C) manual pages refer to programming capabilities available beginning with Issue 4.1 of the C Programming Language Utilities.

Administrators should note the following: if you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight time is starting or ending), and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

NAME

environ – user environment

DESCRIPTION

An array of strings called the "environment" is made available by *exec*(2) when a process begins. By convention, these strings have the form "name=value". The following names are used by various commands:

HOME The name of the user's login directory, set by *login*(1) from the password file (see *passwd*(4)).

PATH The sequence of directory prefixes that *sh*(1), *time*(1), *nice*(1), *nohup*(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (:). *login*(1) sets **PATH=:/bin:/usr/bin**. (For more detail, see the "Execution" section of the *sh*(1) manual entry.)

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as *mm*(1) or *vi*(1), which may exploit special capabilities of that terminal.

TZ Time zone information. The simplest format is **xxx***n***zzz** where **xxx** is the standard local time zone abbreviation, *n* is the difference in hours from GMT (Greenwich Mean Time), and **zzz** is the abbreviation for an alternate time zone (usually the daylight-saving local time zone), if any; for example,

TZ="EST5EDT"

The most complex format allows you to specify the difference in hours of the alternate time zone from GMT and the starting day and time and ending day and time for using this alternate time zone. For example, in 1985 the complex format corresponding to the above simple example is:

TZ="EST5:00:00EDT4:00:00;118/2:00:00,300/2:00:00"

When the above complex format is used, it must be surrounded by double quotes. For more details, see *ctime*(3P) and *timezone*(4).

In the POSIX environment, the format and interpretation of this environment variable is slightly different; see *strftime*(3C).

When a program or a command begins execution in the POSIX environment, the international environment is set to the C *locale*, which corresponds to the traditional UNIX System environment. Programs and commands can change the *locale*, typically to values specified in the following environment variables. For each *locale* category, the corresponding environment variable value is used. If the environment variable value is nonexistent, null, or set to an invalid environment, the *locale* specified in the LANG environment variable is used. If neither LANG or the category-specific environment variable exists, is non-null, or identifies a valid *locale*, then the environment is not changed. For further information, see *locale*(5P).

LC_CTYPE In a POSIX environment, this environment variable specifies the *locale* to be selected for character classification and conversion information. The *locale* is used by commands (such as *cat*(1), *ed*(1), *sort*(1), etc.) to classify characters as alphabetic, printable, uppercase, etc., and to convert characters to uppercase or lowercase, (see *ctype*(3C)).

The *locale* data for this category is created using the *chrtbl*(1M) command (see also *locale*(5P)).

LC_NUMERIC
In a POSIX environment, this environment variable specifies the *locale* to be selected for numeric editing. This *locale* defines the decimal delimiter recognized by the *atof*(3C), *gcvt*(3C), *strtod*(3C), *scanf*(3C), and *printf*(3C) routines.

For information on how to create data for this category, see *locale*(5P) and *localeconv*(3P).

LC_MONETARY
In a POSIX environment, this environment variable specifies the *locale* to be selected for monetary editing, see *localeconv*(3P).

For information on how to create data for this category, see *locale*(5P).

LC_TIME In a POSIX environment, this environment variable specifies the *locale* to be selected for string formatters in date and time editing. The *locale* information is used by the *date*(1) command and by the *strftime*(3P) routines (see *ctime*(3P)).

For information on how to create this locale, see *locale*(5P) and *cftime*(3C).

LANG A value to be used for setting the *locale* if no environment variable corresponding to the individual *locale* categories is set, as defined above.

In the regular *System V* environment, the following environment variables affect the international environment:

CFTIME The default format string to be used by the *date*(1) command and the **ascftime**( ) and **cftime**( ) routines (see *ctime*(3P)). If **CFTIME** is not set or is null, the default format string specified in the **/lib/cftime/***LANGUAGE* file (if it exists) is used in its place (see *cftime*(4)).

CHRCLASS A value that corresponds to a file in **/lib/chrclass** containing character classification and conversion information. This information is used by commands (such as *cat*(1), *ed*(1), *sort*(1), etc.) to classify characters as alphabetic, printable, uppercase, etc., and to convert characters to uppercase or lowercase.

When a program or command begins execution, the tables containing this information are initialized based on the value of **CHRCLASS**. If **CHRCLASS** is nonexistent, null, set to a value for which no file exists in **/lib/chrclass**, or errors occur while reading the file, the ASCII character set is used. During execution, a program or command can change the values in these tables by calling the **setchrclass()** routine. For more detail, see *ctype*(3C).

These tables are created using the *chrtbl*(1M) command.

LANGUAGE A language for which a printable file by that name exists in **/lib/cftime.** This information is used by commands (such as *date*(1), *ls*(1), *sort*(1), etc.) to print date and time information in the language specified.

If **LANGUAGE** is nonexistent, null, set to a value for which no file exists in **/lib/cftime**, or errors occur while reading the file, the last language requested will be used. (If no language has been requested, the language **usa_english** is assumed.) For a description of the content of files in **/lib/cftime**, see *cftime*(4).

Further names may be placed in the environment by the *export* command and "name=value" arguments in *sh*(1), or by *exec*(2). It is unwise to conflict with certain shell variables that are frequently exported by **.profile** files: **MAIL, PS1, PS2, IFS** (see *profile*(4)).

SEE ALSO

exec(2), ctime(3P), ctype(3C), cftime(4), passwd(4), profile(4), timezone(4), locale(5P).

cat(1), chrtbl(1M), date(1), ed(1), env(1), ls(1), login(1), nice(1), nohup(1), sh(1), sort(1), time(1), vi(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

mm(1) in the *DOCUMENTER'S WORKBENCH Software Release 2.0 Technical Discussion and Reference Manual.*

NOTES

References to the *cftime*(4), *ctime*(3P), and *ctype*(3C) manual entries refer to programming capabilities available beginning with Issue 4.1 of the C Programming Language Utilities.

Administrators should note the following: If you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight time is starting or ending) and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

locale − define and set international environment

DESCRIPTION

A *locale* is made up from one or more categories. Each category is identified by its name, and controls specific aspects of the behavior of components of the system. Category names correspond to the following environment variable names.

LC_CTYPE Affects the behavior of the character handling functions.

LC_TIME Affects the behavior of the *strftime* function.

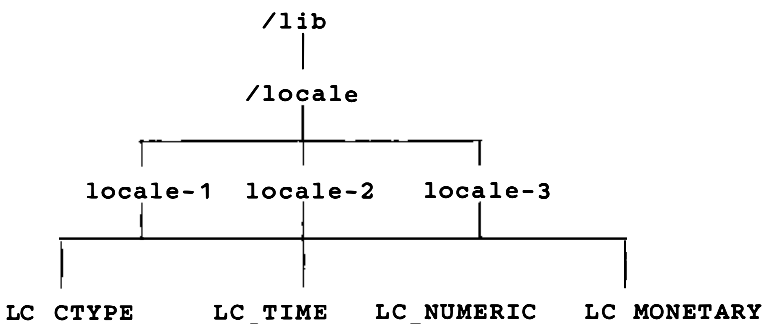
LC_NUMERIC Affects the decimal-delimiter character for the formatted input/output functions and the string conversion functions, as well as the non-monetary formatting information returned by the *localeconv* function.

LC_MONETARY Affects the monetary formatting information returned by the *localeconv* function.

Programs compiled with the **−Xp** option can use the *setlocale* function to modify the environment. When the program starts, the environment is set to the **C** *locale*, which corresponds to the traditional UNIX System environment. Programs can modify this environment by using the *setlocale* function.

If so directed by the program, the values of the above environment variables will be used to set the environment. The value assigned to the environment variable **LANG** will be used as the value for any of the above variables for which no valid value is assigned. If **LANG** is set to a valid value, and none of the above variables are set, then the entire environment will be set to the value indicated by **LANG**.

The information that defines a specific *locale* must be stored in data files on the system. The information for each category is stored in a file with a name corresponding to the environment variable names. The default location is within a directory under **/lib/locale**. The name of the directory is the name of the *locale*:



Creating a Locale

The following steps are used to create the *locale* information.

*locale*s installed under **/lib/locale** should be viewed as "public" *locale*s; all others should be considered private. Installation procedures are the same for both private and public *locale*s. Only the system administrator should be able to create, modify, or delete public *locale*s.

As a first step, create a directory with the desired name of the *locale* within **/lib/locale** (or, in case of a private *locale*, the appropriate directory). Then, the individual categories should be created as described in the following sections.

LC_CTYPE

The information in the **LC_CTYPE** file is generated via the *chrtbl* utility. After executing the *chrtbl* utility, the generated data file must be copied or moved to the *locale* directory and given the name of **LC_CTYPE.** As an example, assume that the name or the desired *locale* is **fr_CH.8859**, and the **chrclass** value in the character classification table is **swiss**, then the following steps should be performed:

$chrtbl *sourcename*
$cp swiss /lib/locale/fr_CH.8859/LC_CTYPE

LC_TIME

The information in the **LC_TIME** file is in text format as described in *cftime*(4). Instead of being placed in the **/lib/cftime** directory, the file must be given the name of **LC_TIME** and placed under the *locale* directory.

Example:

```
Jan
Feb
 ...
Dec
January
February
 ...
December
Sun
Mon
 ...
Sat
Sunday
Monday
 ...
Saturday
%H:%M:%S
%m/%d/%y
%a %b %d %T %Z %Y
AM
PM
```

LC_NUMERIC

The information in the **LC_NUMERIC** file is in text format. Each line in the text file contains a keyword and a value, separated by a space or a tab. Lines starting with a # are ignored. The following keywords are recognized:

LC_NUMERIC This keyword must be the first in the file.

decimal_point The value is the character to be used as decimal delimiter, enclosed in quote marks.

thousands_sep The value is the character used as thousands separator.

grouping The value is a string of semicolon-separated numbers, as described in *localeconv*(3P).

END LC_NUMERIC
This keyword must be the last in the file.

Example:

```
LC_NUMERIC
decimal_point    "."
thousands_sep    " "
grouping         3;3;0
END LC_NUMERIC
```

LC_MONETARY

The information in the **LC_MONETARY** file is in text format. Each line in the text file contains a keyword and a value, separated by a space or a tab. Lines starting with a # are ignored. For a detail definition of the values, see *localeconv*(3P). The following keywords are recognized:

LC_MONETARY This keyword must be the first in the file.

int_curr_symbol The value is the four-character string to be used as international currency symbol, enclosed in quote marks.

currency_symbol The value is the character used as currency symbol.

mon_decimal_point The value is the decimal point used to format monetary values.

mon_thousands_sep The value is the separator used to format monetary values.

mon_grouping The value is a string of semicolon-separated numbers, as described in *localeconv*(3P).

positive_sign The string used to indicate a value for a non-negative formatted monetary quantity.

negative_sign The string used to indicate a negative-valued formatted monetary quantity.

int_frac_digits The number of fractional digits (those after the decimal point) to be displayed in an internationally formatted monetary quantity.

frac_digits The number of fractional digits (those after the decimal point) to be displayed in a formatted monetary quantity.

p_cs_precedes Set to 1 or 0 if the **currency_symbol** respectively precedes or succeeds the value for a non-negative formatted monetary quantity.

p_sep_by_space Set to 1 or 0 if the **currency_symbol** respectively is or is not separated by a space from the value for a non-negative formatted monetary quantity.

n_cs_precedes Set to 1 or 0 if the **currency_symbol** respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

p_sign_posn Set to a value indicating the positioning of the **positive_sign** for a non-negative formatted monetary quantity.

n_sign_posn Set to a value indicating the positioning of the **negative_sign** for a negative formatted monetary quantity.

END LC_MONETARY
This keyword must be the last in the file.

Example:

```
LC_MONETARY
int_curr_symbol  "USD "
currency_symbol  "$"
mon_decimal_point        "."
mon_thousands_sep        " "
mon_grouping     3
negative_sign    "CR"
int_frac_digits 2
frac_digits              2
p_cs_precedes    0
p_sep_by_space   1
n_cs_precedes    0
n_sep_by_space   1
n_sign_posn              1
END LC_MONETARY
```

Locale Naming Conventions and Usage

X/Open recommends that *locale* names follow a certain convention. The recommended format is:

language[_territory][.[codeset]][@modifier]

where:

language Indicates the language area, e.g., Fr (for French).

territory Indicates the geographical area, e.g., CH (for Switzerland), which controls, for example, monetary editing rules.

codeset Indicates the used code set, e.g., 8859.

modifier Can be used to distinguish between otherwise identical names (for instance between two different collation sequences).

Example:

```
$LANG=Fr_CH.8859
$LC_COLLATE=$HOME/mylocale
```

In the above declarations, the default *locale* is a French-Swiss, using the 8859.1 codeset. This is the *locale* chosen for all categories except **LC_COLLATE**, for which a "private" *locale* in the directory *mylocale* is chosen.

FILES

/lib/locale/*	Default directory for *locale* directory structures (* is name of *locale*).
/lib/locale/*/LC_CTYPE	Contains LC_CTYPE information.
/lib/locale/*/LC_COLLATE	Contains LC_COLLATE information.
/lib/locale/*/LC_NUMERIC	Contains LC_NUMERIC information.
/lib/locale/*/LC_MONETARY	Contains LC_MONETARY information.

SEE ALSO

localeconv(3P), setlocale(3P), cftime(4).
chrtbl(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

paths – smail routing database

DESCRIPTION

The *paths* file is the routing database for *smail*. Each line of the file provides routing information to either a host or to a domain. Each line should have either two or three tab-separated (*ascii 0x9*) fields. The format of each line in the *paths* file is:

key route [*cost*]

The *key* field is the key on which searches are performed. Typically, this is either a UUCP host name or a domain name. The *smail* program uses a binary search algorithm when searching the database, so the keys must be sorted in ascending order. Case is ignored when searching, so the keys should be converted to lowercase before sorting (see *lcasep*(8) and *pathproc*(8)).

The *route* field is a *printf* string that details the route that mail to the *key* should take. See the *pathalias* documentation for details.

The optional *cost* field is used by *smail* to determine whether to simply queue outbound UUCP mail, or whether to attempt immediate delivery (usually by invoking *uucico*). If the cost field is present, and the value is at or below *smail*'s *queueing threshold*, then the mail will be queued and an attempt at immediate delivery will be made. This will speed mail delivery between hosts who enjoy an inexpensive *uucp* link, such as a hardwired line or some other low-cost transport medium, while allowing mail sent over more expensive media to accumulate before transmission. If the field is absent, the cost defaults to a value above the *queueing threshold*. The default value for the queueing threshold is equal to the *pathalias* cost DEDICATED+LOW. Thus, direct links with a cost of DEDICATED+LOW or less will see immediate delivery, while the others are queued for later delivery.

Example

Here is a sample *paths* file for a small host, such as a PC, that does not want to maintain complete routing information. It illustrates most of the aspects of the *paths* file. Assume that the PC's name is *mypc* and that it is in domain *.mydomain*. Also assume that it has a dedicated link to a smart host named *bighub*, and that *bighub*'s administrator has given *mypc* permission to use *bighub* as a mail relay. Lastly, assume that *mypc* has a dialed on demand link to another computer named *friend*.

pathalias input:

```
mypc = .mypc.mydomain
mypc friend(DEMAND), bighub(DEDICATED)
smart-host = bighub
```

Here is a *paths* file produced by *pathalias −f inputfile|pathproc*:

```
.mypc.mydomain %s      0
bighub bighub!%s       95
friend friend!%s       300
mypc %s    0
smart-host bighub!%s       95
```

SEE ALSO

lcasep(8), pathproc(8), smail(8) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.
pathalias by Peter Honeyman.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

intro — introduction to special files

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals and UNIX system device drivers. STREAMS [see *intro*(2)] software drivers, modules, and the STREAMS-generic set of *ioctl*(2) system calls are also described.

For hardware-related files, 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 UNIX system device driver are discussed where applicable.

Disk device file names are in the following format:

/dev/{r}dsk/#s#

where **r** indicates a raw interface to the disk, the first **#** indicates the drive number, and the second **#** indicates the section number of the partitioned device.

NAME

aha1540 – low-level controller module

DESCRIPTION

The *aha1540* module provides low-level interface routines between the High Performance Disk Driver (see *hpdd*(7)) and the Adaptec AHA-1540, AHA-154XA, and AHA-1640 Intelligent Small Computer System Interface (SCSI) Host Adapters. It can be configured for disk or streaming tape support for one or more host adapter boards, each of which must be the sole initiator on a SCSI bus. The low-level code determines if the adapter is, in fact, present at the configured address (see *disk*(7) and *tape*(7)) and what types of devices are attached to it. It handles simultaneous requests for all devices, overlapping operations where the devices will permit.

For information on configuring a kernel to include the *aha1540* module, refer to the "INTERACTIVE UNIX Operating System Maintenance Procedures."

Board Configuration

The default configurations described in the *Adaptec AHA-1540/1542 User's Manual* or *Adaptec AHA-1640 User's Manual* should be used for standard configurations of the system. If more than one board is to be used in a single system, each must at least occupy a different set of I/O address ranges and use a different DMA channel. Use of a different interrupt level for each board is encouraged, although not required.

SEE ALSO

athd(7), disk(7), hpdd(7), tape(7), tmc8x0(7).
"INTERACTIVE UNIX Operating System Maintenance Procedures" in the *INTERACTIVE UNIX Operating System Guide*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

asy − asynchronous serial port

DESCRIPTION

The *asy* driver is a character device driver written for a front-end module and a common interface module implementation. The *asy* driver supports both the system board serial port and additional serial adapters. Up to 16 serial ports are supported simultaneously. If an adapter for a port is not installed, an attempt to *open* it will fail. The port can be programmed for different speeds (50-38400 baud), for character length, and for parity. Output speed is always the same as input speed. The port behaves as described in *termio*(7).

The asynchronous port is a character-at-a-time device for both input and output. This characteristic both limits the bandwidth which can be achieved over a line and increases the interrupt loading on the central processor. In particular, file transfer programs such as *uucp*(1) may not function well at speeds over 9600 baud.

If the port is opened with the modem control bit present in the minor device number for the port (see below), modem control will be enabled. If enabled, the driver will wait in *open* until Data Carrier Detect (DCD) is present. Once opened, if DCD drops, the driver will return errors on any subsequent *read*s or *write*s of the asynchronous port by the user. If the port is opened as a controlling teletype, signal SIGHUP will be generated to the process which did the *open*.

The baud rates of the serial adapter's programmable baud rate generator do not correspond exactly with system baud rates. In particular, setting B0 will cause a disconnect, i.e., cause the Ready-to-Send (RTS) and Data-Term-Ready (DTR) lines to be dropped, whereas setting the baud rate back to any non-zero value will assert the RTS and DTR lines. Setting EXTA will set 19200 baud, and setting EXTB will set 38400 baud. It is not possible to directly set 2000, 3600, or 7200 baud.

Minor Device Numbers

The asynchronous port is a character device. The low-order 4 bits (bits 0-3) of the minor device number correspond to the asynchronous port. Each port has three devices associated with it:

Device 1. Completely ignores modem control. The device is intended for direct (hardwired) connections, and it allows a simple three-wire connection.

Device 2. Used for dial-in connections, with *open*s blocked until DCD is present.

Device 3. Used for dial-out connections, which allows *open* calls to complete immediately, although the device obeys modem control once a carrier has been established.

Bit 4 indicates the dial-in line, and bit 5 indicates the dial-out line. Thus, minor device 0 corresponds to the directly-connected serial port 0 (or async adapter COM1) with no modem control, while minor device 16 corresponds to the same port for dial-in use with modem control enabled, and minor device 32 is the dial-out device with modem control enabled. Similarly, minor device number 1 corresponds to the directly connected serial port 1 (or async adapter COM2), while minor device 17 corresponds to the same port for dial-in use, and minor device 33 is the dial-out device. All three devices interlock to allow use of a single port for bidirectional communication without requiring any special software. If the incoming line has been successfully *open*ed (due to a carrier being detected), then subsequent attempts to *open* either the direct or dial-out lines will fail and *errno* will be set to EBUSY. If either of the direct or dial-out lines has been *open*ed, then a process attempting to *open* the dial-in line will not complete the *open* regardless of the DCD state as long as the other device remains open. This interlocking allows a standard **/etc/getty** to run on the dial-in line while allowing *uucp* to dial out on the dial-out line whenever the incoming line is not in use.

FILES

/dev/tty0* directly connected device

/dev/ttyd* dial-in device

/dev/acu* dial-out device

SEE ALSO

termio(7).
signal(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

athd – low-level controller module

DESCRIPTION

The *athd* module provides low-level interface routines between the High Performance Disk Driver (see *hpdd*(7)) and the MFM, RLL, or ESDI controllers in an AT-compatible machine. It can be configured for disk support for one or more controller boards. The low-level code determines if the controller is, in fact, present at the configured address (see *disk*(7) and *tape*(7)) and what devices are attached to it. It handles simultaneous requests for all devices, overlapping operations where the devices will permit.

For information on configuring a kernel to include the *athd* module, refer to the "INTERACTIVE UNIX Operating System Maintenance Procedures."

Board Configuration

The default configuration described in the User's Manual that accompanies the controller board should be used for standard configurations of the system. If more than one board is to be used in a single system, each must at least occupy a different set of I/O address ranges and use a different DMA channel. Use of a different interrupt level for each board is required.

SEE ALSO

aha1540(7), disk(7), hpdd(7), tape(7), tmc8x0(7).
"INTERACTIVE UNIX Operating System Maintenance Procedures" in the *INTERACTIVE UNIX Operating System Guide*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

clone – open any minor device on a STREAMS driver

DESCRIPTION

clone is a STREAMS software driver that finds and opens an unused minor device on another STREAMS driver. The minor device passed to *clone* during the open is interpreted as the major device number of another STREAMS driver for which an unused minor device is to be obtained. Each such open results in a separate *stream* to a previously unused minor device.

The *clone* driver consists solely of an open function. This open function performs all of the necessary work so that subsequent system calls [including *close(2)*] require no further involvement of *clone*.

clone will generate an ENXIO error, without opening the device, if the minor device number provided does not correspond to a valid major device, or if the driver indicated is not a STREAMS driver.

CAVEATS

Multiple opens of the same minor device cannot be done through the *clone* interface. Executing *stat(2)* on the file system node for a cloned device yields a different result from executing *fstat(2)* using a file descriptor obtained from opening the node.

SEE ALSO

log(7).
STREAMS Programmer's Guide.

NAME

console — console interface

DESCRIPTION

The console provides the operator interface to the computer.

The file **/dev/console** is the system console, and refers to the video adaptor and keyboard. This special file implements the features described in *termio*(7), *display*(7), and *keyboard*(7).

FILES

/dev/console
/dev/vtmon
/dev/vt00

SEE ALSO

termio(7), display(7), keyboard(7).

NAME

cpyrt — OEM copyright driver

DESCRIPTION

The kernel *cpyrt* driver is used to print OEM and VAR copyright messages on the console at system initialization. The "driver" consists of only a start routine, which is called after all the other device drivers have run their initialization routines. A driver that needs to display an OEM copyright notice on boot up should put a pointer to the message string in the first empty entry in the global array during its initialization routine. The *cpyrt* driver will then write out the strings using *printf* ("%s\n\n"). If there is more than one driver from a given OEM, the initialization code should check the filled slots and compare the copyright strings, so that a given message only appears once. It is imperative that no driver write past the end of the array. The following code fragment shows how a driver might use this capability:

```
extern int      max_copyrights;
extern char     *oem_copyrights[];

     int index;
     char *msgptr = "YOUR MESSAGE HERE";

     for (index = 0; index < max_copyrights; index++) {
               if ( !oem_copyrights[index] ) {
               /* empty slot found, put your pointer here */
                    oem_copyrights[index] = msgptr;
                    break;
               }
               if ( !strcmp(msgptr, oem_copyrights[index]) )
                    break;  /* msg already there */
     }
```

The default number of entries in the array is 10. This number is defined in the **space.c** file, and may be increased if desired. The array is initialized to all zeros (NULL).

FILES

/etc/conf/pack.d/cpyrt/Driver.o
/etc/conf/pack.d/cpyrt/space.c
/etc/conf/sdevice.d/cpyrt

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

cram — CMOS RAM interface

DESCRIPTION

The *cram* driver provides an interface to the 64 bytes of battery backed-up RAM. This memory contains information such as diagnostics and configuration information. For details see the appropriate hardware technical reference manual for your computer.

Ioctl Calls

CMOSREAD — This call is used to read the contents of one of the CMOS RAM locations. The argument to the *ioctl* is the address of a buffer of two unsigned characters, the first of which is the address to be read. The *ioctl* will fill in the second byte with the data. An address less than 0xe or greater than 0x3f will result in an error, with *errno* set to ENXIO.

CMOSWRITE — This call is used to write a value into one of the CMOS RAM locations. The argument to the *ioctl* is the address of a buffer of two unsigned characters, the first of which is the address and the second of which is the value to write at that address. An address less than 0xe or greater than 0x3f will result in an error, with *errno* set to ENXIO. Note that only the superuser may open the CMOS RAM device for writing, and that the CMOSWRITE *ioctl* will fail for any user other than the superuser.

FILES

/dev/cram

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

disk – random access bulk storage

DESCRIPTION

The secondary storage devices used by the system are fixed disks and diskettes. Disks are high-speed rotating magnetic media. There are several platters in a fixed disk and one in a diskette. Each platter provides up to two surfaces on which data can be stored; on each such surface, data is written in concentric rings known as *tracks*. Each set of parallel tracks on all the usable surfaces of a drive is considered to be a *cylinder*. Each track is divided into several *sectors*, each of which can hold the same number of data bytes (frequently 512). A sector is usually the smallest unit which can be transferred to or from the disk when using the *raw* disk devices. However, the system allows *read* or *write* operations of any size to or from any location on the disk when using the *buffered* disk devices. The special files for raw (unbuffered) disk devices are found in the directory **/dev/rdsk**, and the special files for the buffered disk devices are found in the directory **/dev/dsk**.

High Performance Disk Driver

Access to all fixed disk devices under the INTERACTIVE UNIX Operating System is provided by the High Performance Disk Driver (HPDD). This is a collection of routines that implements the kernel's standard block device entry points. Actual access to various types of disks and controllers is handled by a set of low-level driver modules, one for each distinct type of controller. Low-level driver modules exist for AT-compatible fixed disk controllers, PS/2 Model 80 ESDI and ST-506 controllers, several SCSI host adapters, and for a RAM disk. The low-level modules are responsible for initializing a particular type of controller and providing information as to the disks attached to it, performing *read* and *write* operations given physical sector and memory addresses, and handling controller-specific details of certain *ioctl* operations. The HPDD is responsible for all request scheduling, bad-block mapping, virtual address translation, error reporting, and management of fixed disk partitions. It also adjusts the manner in which I/O requests are presented to low-level modules based on the capabilities of the underlying hardware. For example, it is possible to cause the HPDD to not generate low-level transfer requests that span multiple cylinders or more than a certain number of sectors, or to cause it to make explicit seek requests, as required.

The low-level modules supported in a given system, as well as the mapping of device special files to physical disks, is specified when the system kernel is configured. Up to 8 physical disk devices can be supported on multiple controllers. See the section "High Performance Disk Drivers" in the "INTERACTIVE UNIX Operating System Maintenance Procedures" for more information.

Logical Disks

It is often useful to partition fixed physical disks into smaller sections, each of which may (although it need not) contain a UNIX System *file system*. A file system is a collection of data structures on the disk created by the *mkfs*(1M) program and used by the UNIX System to store user files. The disk device driver can divide a physical disk into smaller *logical disks* or *partitions*. The partitioning occurs at two levels. The BIOS provides for up to four distinct operating system partitions and records this in the first sector on the disk in the *fdisk* table. The HPDD can provide access to any of these partitions and can also further divide the UNIX Operating System's partition. The partitions are defined during system installation (for the primary drive) or when an additional drive is first added to the system. See the section "Adding a Second Fixed Disk" in the "INTERACTIVE UNIX Operating System Installation Instructions" for information.

Diskettes are not partitioned as are fixed disks. There are up to four available "partitions" on each diskette. The first of these spans the entire diskette. The second specifies all but the first cylinder. The third covers all but the first two cylinders. The fourth is undefined. Normally, the full diskette is used for data storage. This may be unstructured (as in a *cpio*(1) file) or may contain a file system. The second and third forms are used primarily for *bootable* diskettes, where the system bootstrap program is written in the first and second cylinders and the root file system is contained in the remainder of the diskette.

Disk Device Special Files

Each fixed disk partition has a specific device special file associated with it. There are rigid rules for the construction of their names. The device names are of the forms *c*i*d*j**p**k and *c*i*d*j**s**k, where **i** is the (0-based) controller number, **j** is the (0-based) drive number on the controller, and **k** is the partition number (0-f) on that drive. Partition names with a *p* indicate *fdisk* table partitions, while *s* partitions are UNIX System sub-partitions. For drives on the primary controller, the *c0d* portion of the name is dropped, yielding a name like *0s1* for the **root** file system's partition. Thus, the special file for the fourth partition on the third drive attached to the second controller on the system would be **/dev/[r]dsk/c***1* **d***2***s***3* (remember all numbers are 0-based). Partition **p0** always describes the entire physical disk; **s0** is the entire UNIX System part.

The device special file names for diskettes also follow rigid naming conventions. These names are of the form *f***i**{*dq*}**j**{*sd*}[*t*|*b*] where **i** is the (0-based) number of the diskette (floppy) drive, **d** indicates a *double-density* diskette, and **q** indicates a *quad- (high-) density* diskette (either **d** or **q** must be present) having **j** sectors per track. **s** indicates a *single-sided* diskette and **d** indicates a *double-sided* diskette (either **s** or **d** must be present); **t** (if present) indicates the entire (or total) diskette. Names not ending in **t** indicate all but the first cylinder; names ending with a **b** do not include the first two cylinders. Thus, the special file **/dev/rdsk/f0q15dt** refers to all of a high density dual-sided 5 ¼-inch diskette in the first drive accessed in unbuffered (raw) mode, and **/dev/dsk/f1q18d** refers to the file system portion of a bootable high density dual-sided 3 ½-inch diskette in the second drive accessed in buffered mode.

Bad Sector Re-mapping

NOTE: Bad sector re-mapping is not supported on diskettes.

Information recorded on fixed disk drives is stored in the form of tiny magnetic domains in an oxide coating on each surface of the disk.

Since this coating is applied mechanically, certain areas of the coating may be flawed. Data recorded in a flawed area may be read incorrectly. Several techniques are used to minimize the effects of such surface defects. The *sector headers* written at low-level format time, which contain information about the address (cylinder, head, and sector number) of each sector on the disk, contain *Cyclic Redundancy Check* (CRC) values. These are recomputed when the headers are read. If the computed value fails to match the read value, some number of data bits in the header are known to be incorrect. In addition, the data portion of each sector contains either a CRC or an *Error Correcting Code* (ECC). In addition to spotting defective data, an ECC can also be used to correct some number of erroneous data bits. Thus, by writing and reading all the sectors of a disk prior to storing actual data on it, those sectors that do not reliably hold the recorded data can be found. This process is known as *surface analysis*.

In addition to these *hard* errors (defects that always cause data transfer failures), *soft* defects may also exist. These are areas of the disk that *may* hold data for some period of time, but cannot be depended on for long periods or under marginal electrical conditions. These defects are far less common in practice than hard errors, but they are considerably more difficult to locate.

Drive manufacturers rigorously test their drives under forced marginal conditions for long periods of time. Any areas that are defective are marked in a *Manufacturer's Defect List* which is shipped with the drive (in hardcopy form) and may also be recorded in a known area on the drive. During initialization of a fixed disk under the UNIX Operating System, an attempt is made to provide the system with a logically-contiguous set of sectors that can be considered to be error-free. This is done by noting defective sectors and providing an alternative good sector for each one found to be bad. This process is known as bad sector re-mapping, and it is performed in different ways depending on the type of drive and controller being used.

First, the drive may be formatted. This process rewrites all the sector headers and sets all the data fields to known values. Formatting the drive will reinforce any header data that had been weakened by shock or exposure to magnetic fields during shipment of the drive. In addition, some controllers (like those for SCSI disks) will actually read the manufacturer's defect list and perform bad sector re-mapping on their own. By doing this, they eliminate all defective (or marginal) sectors known to exist when the drive was built. Other manufacturers format their drives in such a way that an extra sector is present in each track. This sector (which is not normally accessible) is given the address of any other sector in the track that is found to be defective. Thus, only a track with more than one defect (very rare) will actually exhibit a defect when the normally-available sectors are used. This is known as sector slipping.

Second, a partial (*read*-only) or complete (*write* all, then *read*) surface analysis is performed. This is done with controller retries and error correction disabled (if possible, depending on the controller) so that soft errors will be detected. All defective sectors located during surface analysis are noted.

Third, the user is asked to enter any known defects. This can be from a printed manufacturer's defect list (unless such defects are handled by the controller as described above) or those found during normal system operation. All the defects obtained from these latter two sources are entered in an *alternate-sectors* table stored on the drive. For each possible entry in the alternates table, a spare sector is reserved (and verified to be good). Any sector listed as bad in the alternates table will not be used. Routines in the HPDD will re-direct requests for such sectors to their good alternates. This is done transparently to the operating system.

If a latent defect emerges during the running of the system (a very rare event, in practice), the **−A** option of *mkpart*(1) can be used to add them to the alternates table after the system has been installed and is running. The data that had been in the sector that is reassigned in this manner is lost. This may mean that from a sector's worth of data in a file up to an entire file system may be corrupted, depending on the location of the defective sector in the file system.

The software bad sector re-mapping scheme is enabled on a per-partition basis. Partitions 0 (the entire UNIX System partition) and any MS-DOS (DOS) partitions (p0−p4) are exempt from re-mapping.

Fixed Disk Layout

The first sector of the disk (sector 0) consists of a data structure called the *fdisk* table. The *fdisk* table describes the partitions on the fixed disk. *fdisk* partitions should not be confused with UNIX System partitions. *fdisk* partitions describe entire operating system areas that may reside on the disk, e.g., UNIX, DOS, and OS/2.

A total of four physical partitions can be described. At least one of the partitions must describe a UNIX System partition. The *fdisk* partition structure and *fdisk* table look like:

```
/*
 * structure to hold the fdisk partition table
 */
struct ipart{
   unsigned char bootid;        /* bootable or not */
   unsigned char beghead;       /* beginning head, sector, cylinder */
   unsigned char begsect;       /* begcyl is a 10-bit number */
   unsigned char begcyl;          /* high 2 bits are in begsect */
   unsigned char systid;        /* OS type */
   unsigned char endhead;       /* ending head, sector, cylinder */
   unsigned char endsect;       /* endcyl is a 10-bit number */
   unsigned char endcyl;          /* high 2 bits are in endsect */
   long reselect;               /* first sector relative to start of disk */
   long numsect;                /* number of sectors in partition */
};
```

```
/* Fdisk Structure
 * structure to hold master boot block in physical sector 0 of the disk
 * note that partitions structure can't be directly included in the structure
 * due to 386 compiler alignment design
 */
struct mboot {                          /* master boot block */
    char  bootinst [BOOTSZ];
    char  parts [FD_NUMPART * sizeof (struct ipart)];
    ushort signature;
};
```

The *relsect* field of the table entry points to the start of the *fdisk* partition. At least one of the partitions must describe a UNIX System partition.

Inside the UNIX partition at logical sector 29 is the UNIX System *pdinfo* table. The *pdinfo* table points to both the *VTOC* and the *alts_table*. Usually, the *pdinfo* and *VTOC* are in the same sector, and the *alts_table* is in the following sector. The *alts_table* is included for backward compatibility and is used only in the absence of an ALTSCTR partition (see *mkpart*(1M)).

The *VTOC* can describe up to 16 UNIX System partitions. See **/usr/include/sys/vtoc.h** for a description of these partitions.

Ioctl Calls Supported

All the standard disk *ioctl* calls defined in the file **/usr/include/sys/vtoc.h** are supported for *all* disk devices. In addition, each low-level driver module may support controller-specific *ioctl*s. See the low-level module descriptions for details. The standard calls supported are:

V_CONFIG — This call is used by *mkpart* to reconfigure the drive, so that the drive configuration matches the parameters specified in the **/etc/partitions** file. The argument to the *ioctl* is the address of one of the following structures, defined in **<sys/vtoc.h>**, containing the new configuration parameters:

```
union io_arg {
    struct {
        ushort   ncyl;       /* number of cylinders */
        unchar   nhead;      /* heads/cylinder */
        unchar   nsec;       /* sectors/track */
        ushort   secsiz;     /* bytes/sector */
    } ia_cd;
}
```

Note that it is not possible to change the sector size on a fixed disk, and that an attempt to do so will result in the *ioctl* failing, with *errno* set to EINVAL.

V_REMOUNT

This call is used to force the driver to re-read the *VTOC* on the next open of the drive. It will fail if any partition other than partition 0 is currently open, since changing the partition table information is potentially disastrous for a process using the partition. This is used by *mkpart* when it changes the *VTOC*, so that the driver will update its internal tables.

V_XGETPARMS

This call is used to get information about the current drive configuration. The argument to the *ioctl* is the address of one of the following structures, defined in **<sys/vtoc.h>**, which will be filled in by the *ioctl*:

```
struct disk_xparms {
   char     dpx_type;             /* Disk type (see below) */
   ulong    dpx_heads;            /* Number of heads */
   ulong    dpx_cyls;             /* Number of cylinders */
   ulong    dpx_sectors;          /* Number of sectors/track */
   ulong    dpx_secsiz;           /* Number of bytes/sector */
                                  /* for this partition: */
   ushort   dpx_ptag;             /* Partition tag */
   ushort   dpx_pflag;            /* Partition flag */
   union    {                     /* value depends on partition # */
            struct {              /* returned for partition 0: */
            ulong dp0_secovhd;    /* Controller's per-sector overhead */
            ulong dp0_rsrvdcyls;  /* # of reserved cylinders from total */
            ulong dp0_intlv;      /* Interleave factor (0 if unknown) */
            ulong dp0_skew;       /* Sector skew factor (per head) */
                                  /* Meaningful only if dp0_intlv == 1 */
                                  /* If intlv == 1 & skew == 0, unknown */
            ulong dp0_ctlflags;   /* Controller-specific flags */
            ulong dp0_pheads;     /* Physical heads */
            ulong dp0_pcyls;      /* Physical cylinders */
            ulong dp0_psectors;   /* Physical sectors/track */
            ulong dp0_psecsiz;    /* Physical bytes/sector */
                } dp0_ctl;
            daddr_t dp1_pstartsec;/* returned for any other partition */
            } dpx_psense;
   daddr_t dpx_pnumsec;           /* Number of sectors */
   };

/* Disk types */
#define      DPT_WINI           1    /* Winchester disk */
#define      DPT_FLOPPY         2    /* Floppy */
#define      DPT_OTHER          3    /* Other type of disk */
#define      DPT_NOTDISK        0    /* Not a disk device */

/* Partition identification tags */
#define      V_UNUSED        0x00    /* Unused partition */
#define      V_BOOT          0x01    /* Boot partition */
#define      V_ROOT          0x02    /* Root filesystem */
#define      V_SWAP          0x03    /* Swap filesystem */
#define      V_USR           0x04    /* Usr filesystem */
#define      V_BACKUP        0x05    /* full disk */
```

```
#define    V_ALTS      0x06   /* alternate sector space */
#define    V_OTHER     0x07   /* non-unix space */
#define    V_ALTTRK    0x08   /* alternate track space */
#define    V_ALTSCTR   0x09   /* alternate sector partition */

/* Partition flag */
#define    V_UNMNT     0x001  /* unmountable partition */
#define    V_RONLY     0x010  /* read only partition */
#define    V_OPEN      0x100  /* partition open */
#define    V_VALID     0x200  /* partition valid to use */
```

For the fixed disk driver, the disk type will always be **DPT_WINI.** Since the structure returned by **V_XGETPARMS** is the same for both the diskette and fixed disk drivers, programs may be written to understand either one.

V_FORMAT

This call is used to format tracks on a disk. The argument passed to the *ioctl* is the address of one of the following structures, defined in **<sys/vtoc.h>**, containing the starting track, number of tracks, and interleave factor:

```
union io_arg {
    struct {
            ushort   start_trk;     /* first track */
            ushort   num trks;      /* number of tracks to format */
            ushort   intlv;         /* interleave factor */
    } ia_fmt;
}
```

Note that the file descriptor argument to the *ioctl* must refer to the character (raw) special device for the desired drive, and the file must have been opened in exclusive mode, i.e., **O_EXCL.**

Also note that a drive will only accept one of either **V_FORMAT** or **V_FMTDRV** (see next paragraph), depending on the capabilities of the controller. Typically, SCSI disks require **V_FMTDRV,** while MFM, RLL, and ESDI controllers need **V_FORMAT.**

V_FMTDRV

This call is used to format the entire disk. The argument passed to the *ioctl* is the desired interleave factor.

Note that the file descriptor argument to the *ioctl* must refer to the character (raw) special device for the desired drive, and the file must have been opened in exclusive mode, i.e., **O_EXCL.**

V_PDLOC

This call is used to return the starting sector number of the *pdinfo* structure. This information is passed back in the return argument as *arg*.

V_RETRYCTL

This call is used to direct the driver to turn retry and error correction logic on or off. The argument passed will be one of:

```
V_RETRY_ON          /* turn retry/error correction on */
V_RETRY_OFF         /* turn retry/error correction off */
```

For each disk device, the initial state for this flag is *on*.

Configuration Parameters

The configuration parameters are found in the **cf.d/mtune** file. The parameters that apply to the High Performance Disk Driver (HPDD) are:

DISK_DRQBLOX	1500 (default)
RAMSIZE	384 (default)

Error Messages

The HPDD has a well-defined set of error conditions. Each of the low-level drivers is responsible for translating its error conditions into the appropriate HPDD error condition. The normal course of action taken for errors is to retry the operation if the controller is set up for retrys. If the operation succeeds upon retry, then no error is reported. If after 10 retrys the operation still fails, then an error message is printed on the console. The possible error messages are:

Message	*Action Taken*
Logic Problem. NO ERROR FOUND	PANIC
Data Address Mark not found	RETRY
Track 0 not found (unable to recalibrate)	PANIC
Write fault on drive	PANIC
Drive is not ready	NORETRY
Controller will not come ready	RETRY
Seek will not complete	PANIC
Seek error (wrong cylinder found)	RETRY
No Index signal found	RETRY
Medium is write-protected	NORETRY
Medium is not present in drive	NORETRY
Error found in sector ID field	RETRY
Sector not found	RETRY
Uncorrectable data error in sector	RETRY
Sector or track was marked bad	QUIET
Error during FORMAT operation	RETRY
Illegal or erroneous command	RETRY
Controller error or failure	PANIC
Controller command Aborted	RETRY
Drive is still seeking	RETRY
Medium has been changed in drive	NORETRY
Attempt to do I/O past end of drive	NORETRY
Command Timeout	PANIC
Unable to get valid drive configuration	RETRY
Undetermined error	RETRY
EOF/EOM Detected	NORETRY & QUIET

SEE ALSO

fdisk(1M), mkfs(1M), mkpart(1), mount(1M), intro(7).
ioctl(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.
"INTERACTIVE UNIX Operating System Maintenance Procedures" and "INTERACTIVE UNIX Operating System Installation Instructions" in the *INTERACTIVE UNIX Operating System Guide*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

display – system console display

DESCRIPTION

The system console (and user's terminal) is composed of two separate pieces: the keyboard (see *keyboard*(7)) and the display. Because of their complexity, and because there are three possible display interfaces (monochrome, color graphics, and enhanced graphics adapters), they are discussed in separate manual entries.

The display normally consists of 25 lines of 80 columns each; 40-column lines are also supported by the color/graphics adapter, and 43 lines of 80-columns each are supported by the enhanced graphics adapter. Writing characters to the console or one of its virtual screens (**/dev/console** or **/dev/vt***xx*) has an effect which depends on the characters. All characters written to **/dev/console** are first processed by the terminal interface (see *termio*(7)). For example, mapping new-line characters to carriage return plus new-line, and expanding tabs to spaces, will be done before the following processing:

x Where *x* is not one of the following, displays *x*.

BEL Generates a *bell* (audible tone, no modulation).

CR Places the cursor at column 1 of the current line.

LF, VT Places the cursor at the same column of the next line (scrolls if the current line is line 25).

FF Clears the screen and places the cursor at line 1, column 1.

BS Depends on the previous character: if an _ (underscore), see below; otherwise, if the cursor is not at column 1, it is moved to the left one position on the same line. If the cursor is at column 1, it is not moved.

_BS*x* Sets the *underscore* attribute for the character *x* to be displayed. The *underscore* attribute for the color/graphics adapter is a red background with a white foreground.

ESC*x* Where *x* is *any* of the 256 possible codes (except **c**, **[**, and **H**), displays that value uninterpreted. This is useful for utilizing the full set of graphics available on the display. Note again that the characters are processed through the terminal interface prior to this escape sequence. Therefore, to get some of the possible 256 characters, it is necessary that the character not be postprocessed. The easiest way to accomplish this is to turn off OPOST in the *c_oflag* field (see *termio*(7)); however, this may have other side effects.

The display can be controlled by means of ANSI X3.64 *escape sequences*, which are specific sequences of characters, preceded by the ASCII character ESC. The escape sequences, which work on either the monochrome, color graphics, or enhanced graphics adapter, are the following:

ESCc Clears the screen and places the cursor at line 1, column 1.

ESC H	Sets a tab stop at the current cursor position. This escape sequence will be ignored if a tab is already set.
ESC Q *n* '*string*'	Defines the function key *n* with *string*. The string delimiter ' may be any character not in *string*. Function keys are numbered 0 through 11 (F1 = 0, F2 = 1, etc.).
ESC[*n* @	Insert character—inserts *n* blanks at the current cursor position.
ESC[*n* `	Horizontal Position Absolute – moves active position to column given by *n*.
ESC[*n* A	Cursor up – moves the cursor up *n* lines (default: *n*=1).
ESC[*n* a	Horizontal Position Relative – moves active position *n* characters to the right (default: *n*=1).
ESC[*n* B	Cursor down – moves the cursor down *n* lines (default: *n*=1).
ESC[*n* C	Cursor right – moves the cursor right *n* columns (default: *n*=1).
ESC[*n* D	Cursor left – moves the cursor left *n* columns (default: *n*=1).
ESC[*n* d	Vertical Position Absolute – moves active position to line given by *n*.
ESC[*n* E	Cursor next line – moves the cursor to column 1 of the next line, then down *n*−1 lines (default: *n*=1).
ESC[*n* e	Vertical Position Relative – moves the active position down *n* lines (default: *n*=1).
ESC[*n* F	Cursor previous line – moves the cursor to column 1 of the current line, then up *n* lines (default: *n*=1).
ESC[*n* G	Cursor horizontal position – moves the cursor to column *n* of the current line (default: *n*=1).
ESC[*n* ; *m* H	Position cursor – moves the cursor to column *m* of line *n* (default: *n*=1, *m*=1).
ESC[*n* ; *m* f	Position cursor – moves the cursor to column *m* of line *n* (default: *n*=1, *m*=1).
ESC[*n* J	Erase window – erases from the current cursor position to the end of the window if *n*=0, from the beginning of the window to the current cursor position if *n*=1, and the entire window if *n*=2 (default: *n*=0).
ESC[*n* K	Erase line – erases from the current cursor position to the end of the line if *n*=0, from the beginning of the line to the current cursor position if *n*=1, and the entire line if *n*=2 (default: *n*=0).
ESC[*n* L	Insert line – inserts *n* lines at the current cursor position (default: *n*=1).

ESC[*n* M	Delete line – deletes *n* lines starting at the current cursor position (default: *n*=1).
ESC[*n* P	Delete character – deletes *n* characters from a line starting at the current cursor position (default: *n*=1).
ESC[*n* S	Scroll up – scrolls the characters in the current window up *n* lines. The bottom *n* lines are cleared to blanks (default: *n*=1).
ESC[*n* T	Scroll down – scrolls the characters in the current window down *n* lines. The top *n* lines are cleared to blanks (default: *n*=1).
ESC[*n* X	Erase character – erases *n* character positions starting at the current cursor position (default: *n*=1).
ESC[*n* Z	Cursor Backward Tabulation – moves active position back *n* tab stops.
ESC[2h	Locks the keyboard and ignores keyboard input until unlocked. Characters are not saved.
ESC[2i	Sends the screen to the host. The current screen display is sent to the application.
ESC[2l	Unlocks the keyboard. Re-enables keyboard input.
ESC[? 7 l	Sets the no-wrap flag, causing characters typed after column 80 to remain at column 80.
ESC[*n* g	Delete tab stop – removes the tab stop at the current cursor position if *n*=0, removes all tab stops if *n*=3 (default: *n*=0).

ESC[*Ps* ; *Ps*; . . . m

Character attributes – each *Ps* is one of the following characters; multiple characters are separated by semicolons. These parameters apply to successive characters being displayed, in an additive manner (e.g., both bold and underscoring can be selected). Only the parameters through 7 apply to the monochrome adapter; all parameters apply to the color/graphics adapter and the enhanced graphics adapter (default: *Ps*=0).

Ps	*Meaning*		
0	all attributes off (normal display) (white foreground with black background)		
1	bold intensity		
4	underscore on (white foreground with red background on color)		
5	blink on		
7	reverse video		
30	black	(gray)	foreground
31	red	(light red)	foreground
32	green	(light green)	foreground
33	brown	(yellow)	foreground
34	blue	(light blue)	foreground
35	magenta	(light magenta)	foreground
36	cyan	(light cyan)	foreground
37	white	(bright white)	foreground
40	black	(gray)	background
41	red	(light red)	background
42	green	(light green)	background
43	brown	(yellow)	background
44	blue	(light blue)	background
45	magenta	(light magenta)	background
46	cyan	(light cyan)	background
47	white	(bright white)	background
ESC[3 *Cm*	Set foreground color *C* where *C* represents the color defined in Table 1 below.		
ESC[4 *Cm*	Set backgound color *C* where *C* represents the color defined in Table 1 below.		
ESC[8 *m*	sets blank (non-display)		
ESC[10 *m*	selects the primary font		
ESC[11 m	selects the first alternate font; lets ASCII characters less than 32 be displayed as ROM characters		
ESC[12 *m*	selects a second alternate font; toggles high bit of extended ASCII code before displaying as ROM characters		
ESC[=*cF*	Set normal foreground color to *c* where *c* represents a color defined in Table 2 below.		
ESC[=*cG*	Set normal background color to *c* where *c* represents a color defined in Table 1 below.		
ESC[=*cH*	Set reverse foreground color to *c* where *c* represents a color defined in Table 2 below.		
ESC[=*cI*	Set reverse background color to *c* where *c* represents a color defined in Table 2 below.		
ESC[=*cJ*	Set graphic foreground color to *c* where *c* represents a color defined in Table 2 below.		
ESC[=*cK*	Set graphic background color to *c* where *c* represents a color defined in Table 2 below.		
ESC[=*cA*	Set the overscan (border) color to *c* where *c* represents a color defined in Table 2 below.		

TABLE 1	
C	*Color*
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

TABLE 2			
C	*Color*	*C*	*Color*
0	Black	8	Grey
1	Blue	9	Lt. Blue
2	Green	10	Lt. Green
3	Cyan	11	Lt. Cyan
4	Red	12	Lt. Red
5	Magenta	13	Lt. Magenta
6	Brown	14	Yellow
7	White	15	Lt. White

Note that for character attributes 30-37, the color selected for foreground will depend on whether the *bold intensity* attribute (1) is currently on. If not, the first color listed will result; otherwise the second color listed will result.

Similarly, for character attributes 40-47, the color selected for background will depend on whether the *blink* attribute (5) is currently on. If the *blink* attribute is not on, then the first color listed will result. If the *blink* attribute is on, then the second color listed will result.

Ioctl Calls

The following *ioctl* calls can be used to change the display used for the video monitor:

SWAPMONO
: This call selects the monochrome adapter as the output device for the system console.

SWAPCGA
: This call selects the color/graphics adapter as the output device for the system console.

SWAPEGA
: This call selects the enhanced graphics adapter as the output device for the system console.

SWAPVGA
: This call selects the video graphics array adapter as the output device for the system console.

The following *ioctl* call may be used to obtain more information about the display adapter currently attached to the video monitor:

CONS_CURRENT
: This call returns the display adapter type currently attached to the video monitor. The return value can be one of: MONO, CGA, EGA, or VGA.

The following *ioctl* calls may be used to switch display modes on the various video adapters:

SW_B40x25
: This call selects 40x25 (40 columns by 25 rows) black and white text display mode. It is valid only for CGA, EGA, and VGA devices.

SW_C40x25
: This call selects 40x25 (40 columns by 25 rows) color text display mode. It is valid only for CGA, EGA, and VGA devices.

SW_B80x25
: This call selects 80x25 (80 columns by 25 rows) black and white text display mode. It is valid only for CGA, EGA, and VGA devices.

SW_C80x25
: This call selects 80x25 (80 columns by 25 rows) color text display mode. It is valid only for CGA, EGA, and VGA devices.

SW_BG320
: This call selects 320x200 black and white graphics display mode. It is valid only for CGA, EGA, and VGA devices.

SW_CG320
: This call selects 320x200 color graphics display mode. It is valid only for CGA, EGA, and VGA devices.

SW_BG640
: This call selects 640x200 black and white graphics display mode. It is valid only for CGA, EGA, and VGA devices.

SW_CG320_D
: This call selects EGA support for 320x200 graphics display mode (EGA mode D). It is valid only for EGA and VGA devices.

SW_CG640_E
: This call selects EGA support for 640x200 graphics display mode (EGA mode E). It is valid only for EGA and VGA devices.

SW_EGAMONOAPA
: This call selects EGA support for 640x350 graphics display mode (EGA mode F). It is valid only for EGA and VGA devices.

SW_ENH_MONOAPA2
: This call selects EGA support for 640x350 graphics display mode with extended memory (EGA mode F*). It is valid only for EGA and VGA devices.

SW_CG640x350
: This call selects EGA support for 640x350 graphics display mode (EGA mode 10). It is valid only for EGA and VGA devices.

SW_ENH_CG640
: This call selects EGA support for 640x350 graphics display mode with extended memory (EGA mode 10*). It is valid only for EGA and VGA devices.

SW_EGAMONO80x25
: This call selects EGA monochrome text display mode (EGA mode 7), which emulates support provided by the monochrome adapter. It is valid only for EGA and VGA devices.

SW_ENHB40x25
: This call selects enhanced 40x25 black and white text display mode. It is valid only for EGA and VGA devices.

SW_ENHC40x25
: This call selects enhanced 40x25 color text display mode. It is valid only for EGA and VGA devices.

SW_ENHB80x25
: This call selects enhanced 80x25 black and white display mode. It is valid only for EGA and VGA devices.

SW_ENHC80x25
: This call selects enhanced 80x25 color text display mode. It is valid only for EGA and VGA devices.

SW_ENHB80x43
: This call selects enhanced 80x43 black and white text display mode. It is valid only for EGA and VGA devices.

SW_ENHC80x43
: This call selects enhanced 80x43 color text display mode. It is valid only for EGA and VGA devices.

SW_MCAMODE
: This call reinitializes the monochrome adapter. It is valid only for monochrome adapters.

SW_VGA_B132x25
: This call selects enhanced 132x25 black and white text display mode. It is valid only on Sigma VGA/H adapters with 132 column support.

SW_VGA_C132x25
: This call selects enhanced 132x25 color text display mode. It is valid only on Sigma VGA/H adapters with 132 column support.

SW_VGA_B132x43
: This call selects enhanced 132x43 black and white text display mode. It is valid only on Sigma VGA/H adapters with 132 column support.

SW_VGA_C132x43
: This call selects enhanced 132x43 color text display mode. It is valid only on Sigma VGA/H adapters with 132 column support.

SW_VGAMONOAPA
: This call selects VGA support for 640x350 monochrome graphics display support (VGA mode F+). It is valid only for VGA devices.

SW_VGA_CG640
: This call selects VGA support for 640x350 color graphics display support (VGA mode 10+). It is valid only for VGA devices.

SW_VGA_B40x25
: This call selects VGA support for VGA 40x25 (40 columns by 25 rows) black and white text display mode. It is valid only for VGA devices.

SW_VGA_C40x25
: This call selects VGA support for VGA 40x25 (40 columns by 25 rows) color text display mode. It is valid only for VGA devices.

SW_VGAB80x25
: This call selects VGA support for VGA 80x25 (80 columns by 25 rows) black and white text display mode. It is valid only for VGA devices.

SW_VGAC80x25
: This call selects VGA support for VGA 80x25 (80 columns by 25 rows) color text display mode. It is valid only for VGA devices.

SW_VGAMONO80x25
: This call selects VGA monochrome text display mode (VGA mode 7+). It is valid only for VGA devices.

SW_BG640x480
: This call selects VGA 640x480 black and white graphics display mode (VGA mode 11). It is valid only for VGA devices.

SW_CG640x480
: This call selects VGA 640x480 color graphics display mode (VGA mode 12). It is valid only for VGA devices.

SW_VGA_CG320
: This call selects VGA 320x200 color graphics display mode (VGA mode 13). It is valid only for VGA devices.

Switching to an invalid display mode for a display device will result in an error.

The following *ioctl*s may be used to obtain information about the current display modes:

CONS_GET
: This call returns the current display mode setting for whatever display adapter is being used. Possible return values include:

  DM_B40x25 (0), black and white 40 columns. CGA and EGA only.

DM_C40x25 (1), color 40 columns. CGA and EGA only.

DM_B80x25 (2), black and white 80 columns. CGA and EGA only.

DM_C80x25 (3), color 80 columns. CGA and EGA only.

DM_BG320 (4), black and white graphics 320x200. CGA and EGA only.

DM_CG320 (5), color graphics 320x200. CGA and EGA only.

DM_BG640 (6), black and white graphics 640x200 high-resolution. CGA and EGA only.

DM_EGAMONO80x25 (7), EGA-mono 80x25. EGA only.

DM_ENH_B80x43 (10), EGA enhanced black and white 80x43.

DM_ENH_C80x43 (11), EGA enhanced color 80x43.

DM_CG320_D (13), EGA mode D.

DM_CG640_E (14), EGA mode E.

DM_EGAMONOAPA (15), EGA mode F.

DM_CG640x350 (16), EGA mode 10.

DM_ENHMONOAPA2 (17), EGA mode F with extended memory.

DM_ENH_CG640 (18), EGA mode 10*.

DM_ENH_B40x25 (19), EGA enhanced black and white 40 columns.

DM_ENH_C40x25 (20), EGA enhanced color 40 columns.

DM_ENH_B80x25 (21), EGA enhanced black and white 80 columns.

DM_ENH_C80x25 (22), EGA enhanced color 80 columns.

DM_VGA_C40x25 (23), VGA color 40x25.

DM_VGA_C80x25 (24), VGA color 80x25.

DM_VGAMONO80x25 (25), VGA mode 7+.

DM_BG640x480 (26), VGA black and white graphics 640x480 (VGA mode 11).

DM_CG640x480 (27), VGA color graphics 640x480 (VGA mode 12).

DM_VGA_CG320 (28), VGA color graphics 320x200 (VGA mode 13).

DM_VGA_B40x25 (29), VGA black and white 40x25.

DM_VGA_B80x25 (30), VGA black and white 80x25.

DM_VGAMONOAPA (31), VGA mode F+.

DM_VGA_CG640 (32), VGA mode 10+.

DM_VGA_B132x25 (35), VGA enhanced black and white 132x25.

DM_VGA_C132x25 (36), VGA enhanced color 132x25.

DM_VGA_B132x43 (37), VGA enhanced black and white 132x43.

DM_VGA_C132x43 (38), VGA enhanced color 132x43.

M_MCA_MODE (0xff), monochrome adapter mode.

MCA_GET
: This call returns the current display mode setting of the monochrome adapter. See CONS_GET for a list of return values. If the monochrome adapter is not installed, the call will fail and *erno* will be set to 22 (EINVAL).

CGA_GET
: This call returns the current display mode setting of the color/graphics adapter. See CONS_GET for a list of return values. If the color graphics adapter is not installed, the call will fail and *erno* will be set to 22 (EINVAL).

EGA_GET
: This call returns the current display mode setting of the enhanced graphics adapter. See CONS_GET for a list of return values. If the enhanced graphics adapter is not installed, the call will fail and *erno* will be set to 22 (EINVAL).

VGA_GET
: This call returns the current display mode setting of the video graphics array adapter. See CONS_GET for a list of return values. If the video graphics array adapter is not installed, the call will fail and *erno* will be set to 22 (EINVAL).

The following *ioctl* calls may be used to map the video adapter's memory into the user's data space:

MAPCONS
: This call maps the display memory of the adapter currently being used into the user's data space.

MAPMONO
: This call maps the monochrome adapter's display memory into the user's data space.

MAPCGA
: This call maps the color/graphics adapter's display memory into the user's data space.

MAPEGA
: This call maps the enhanced graphics adapter's display memory into the user's data space.

MAPVGA
: This call maps the video graphics array adapter's display memory into the user's data space.

You can use *ioctl* calls to input a byte from the graphics adapter port or to output a byte to the graphics adapter port. The argument to the *ioctl* uses the **port_io_arg** data structure:

```
struct port_io_arg {
        struct port_io_struc_args[4];
};
```

As shown in the previous example, the **port_io_arg** structure points to an array of four **port_io_struc** data structures. The **port_io_struc** has the following format:

```
struc port_io_struc {
        char dir;               /* direction flag (in vs. out) */
        unsigned short port;    /* port address */
        char data;              /* byte of data */
};
```

You can specify one, two, three, or four of the **port_io_struc** structures in the array for one *ioctl* call. The value of *dir* can be either IN_ON_PORT (to specify a byte being input from the graphics adapter port) or OUT_ON_PORT (to specify a byte being output to the graphics adapter port). *Port* is an integer specifying the port address of the desired graphics adapter port. *Data* is the byte of data being input or output as specified by the call. If you are not using any of the **port_io_struc** structures, load the *port* with 0, and leave the unused structures at the end of the array. Refer to your hardware manuals for port addresses and functions for the various adapters.

The following *ioctl* calls may be used to input or output bytes on the graphics adapter port:

MCAIO This call inputs or outputs a byte on the monochrome adapter port as specified.

CGAIO This call inputs or outputs a byte on the color/graphics adapter port as specified.

EGAIO This call inputs or outputs a byte on the enhanced graphics adapter port as specified.

VGAIO This call inputs or outputs a byte on the video graphics array adapter port as specified.

To input a byte on any of the graphics adapter ports, load *dir* with IN_ON_PORT and load *port* with the port address of the graphics adapter. The byte input from the graphics adapter port will be returned in *data*.

To output a byte, load *dir* with OUT_ON_PORT, load *port* with the port address of the graphics adapter, and load *data* with the byte you want to output to the graphics adapter port.

The following *ioctl*s can be used with either the monochrome, color graphics, or enhanced graphics adapters:

KDDISPTYPE
: This call returns display information to the user. The argument expected is the buffer address of a structure of type **kd_disparam** into which display information is returned to the user. The **kd_disparam** structure is defined as follows:

```
struct kd_disparam {
   long type;                    /* display type */
   char *addr;                   /* display memory address */
   ushort ioaddr[MKDIOADDR];     /* valid I/O addresses */
}
```

Possible values for the type field include:

KD_MONO (0x01)	IBM monochrome display adapter
KD_HERCULES (0x02)	Hercules monochrome graphics adapter
KD_CGA (0x03)	IBM color graphics adapter
KD_EGA (0x04)	IBM enhanced graphics adapter
KD_VGA (0x05)	VGA adapter

The *addr* member indicates the physical address of the display. It will be one of:

MONO_BASE	0xb0000
COLOR_BASE	0xb8000
EGA_BASE	0xa0000

The *ioaddr* member contains the global keyboard/display port list that is used for direct *in*s and *out*s to the screen. When using a VGA display, the result of a KDISPTYPE *ioctl* is the same as with an EGA display.

KIOCSOUND
: This call starts the sound generation. It turns on sound. The argument is the inverse frequency desired. A value of 0 turns off the sound.

KDGETLED
: This call gets the keyboard LED status. The argument is a pointer to a character. The character will be filled with a boolean combination of the following values:

LED_SCR	0x01(flag bit for scroll lock)
LED_CAP	0x04(flag bit for caps lock)
LED_NUM	0x02(flag bit for num lock)

KDSETLED
: This call sets the keyboard LED status. The argument is a character whose value is the boolean combination of the values listed under KDGETLED.

KDMKTONE
: This call generates a fixed length tone. The argument is a 32-bit value, with the lower 16 bits set to the inverse frequency and the upper 16 bits set to the duration (in milliseconds).

KDGKBTYPE
: This call gets keyboard type. The argument is a pointer to a character type. The character will be returned with one of the following values:

KB_84	0x01(84-key keyboard)
KB_101	0x02(101-key keyboard)
KB_OTHER 0x03	

KDADDIO

This call adds the I/O port address to the list of valid video adapter addresses. Argument is an unsigned short type which should contain a valid port address for the installed video adapter.

KDDELIO

This call deletes the I/O port address from the list of valid video adapter addresses. Argument is an unsigned short type which should contain a valid port address for the installed video adapter.

KDENABIO

This call enables *in*s and *out*s to video adapter ports. No argument.

KDDISABIO

This call disables *in*s and *out*s to video adapter ports. No argument.

KDSBORDER

This call sets the screen color border in EGA text mode. The argument is of type character. Each bit position corresponds to a color selection. From bit position 0 to bit position 6, the color selections are respectively blue, green, red, secondary blue, secondary green, and secondary red. Setting the bit position to a logic one will select the desired color or colors.

KDSCROLL

This call is used to set the hardware scrolling feature to be on or off. Most CGA, EGA, and VGA displays provide hardware-assisted scrolling for better output performance. The default is hardware scrolling turned off. A nonzero argument enables scrolling; a zero argument disables it.

KDSETMODE

This call sets the mode of the display driver to the integer argument given. The modes that may be specified are:

KD_TEXT0	0x00
KD_GRAPHICS	0x01
KD_TEXT1	0x02

KD_TEXT is a synonym for KD_TEXT0.

KD_TEXT0 indicates that all text on the display must be written with the *write* system call. The display will automatically be saved and restored on "hot key" screen switches. Upon leaving KD_TEXT0 mode, the display contents will be saved; upon returning to KD_TEXT0 mode, the display will be restored from the saved display. KD_TEXT1 works in the same manner as KD_TEXT0, except that the display is not restored when the user returns to KD_TEXT1 mode.

KD_GRAPHICS mode indicates that the user will have direct control of the display. It will be necessary to map in the display using the KDMAPDISP *ioctl* subsequent to setting KD_GRAPHICS mode to store characters directly onto the display. In this mode, all writes to the display using the *write* system call are ignored. In KD_GRAPHICS mode, the user is responsible for saving and restoring the display on "hot key" switches. This requires that the virtual terminal must be in process (VT_PROCESS) mode prior to setting KD_GRAPHICS mode.

In KD_GRAPHICS mode, there will not necessarily be graphics on the display. When setting KD_TEXT0 or KD_TEXT1 mode, the *ioctl* will fail if the display is mapped in [EIO]. When setting KD_GRAPHICS mode, the *ioctl* will fail if the virtual terminal is not in process (VT_PROCESS) mode [EACCES].

KDGETMODE

This call gets the current mode of the console. It returns an integer argument containing KD_TEXT, KD_TEXT1, or KD_GRAPHICS as defined in the KDSETMODE *ioctl* description.

KDMAPDISP

This call allows one to have direct access to the display and I/O ports. It is especially useful in providing a way to do non-portable but fast graphics on the display. The following structure, defined in **<sys/kd.h>**, is pointed to by the argument to the *ioctl*:

```
struct kd_memloc {
   char     *vaddr;      /* virtual address to map to*/
   char     *physaddr;   /* physical address to map from */
   long     length;      /* size in bytes to map */
   long     ioflg;       /* enable I/O addresses if non-zero */
};
```

The *vaddr* argument is the linear address in the process where the display buffer will appear. This address must be on a page (4K byte) boundary. The *physaddr* argument is the physical address of the screen. It must be between 0xA0000 and 0xC0000. It must also be on a page boundary. The *length* argument is the size of the display buffer that will be mapped in. It must be a multiple of 4K bytes. The *ioflg* argument tells whether (1) or not (0) to enable the global keyboard/display ports for direct access to the I/O ports similar to the KDENABIO and KDDISABIO *ioctl*.

The memory that had existed at address *vaddr* for *length* bytes will be irretrievably deleted, and the current contents of the display buffer will be placed at those locations. It is necessary for the virtual terminal to be in process (VT_PROCESS) mode and for the display device to be in KD_GRAPHICS mode. One way of allocating the virtual memory in the user's address space that will be mapped to the screen is to call *malloc* requesting (*length* + 4096) bytes. Then using the address that *malloc* returns, round it up to the next page (4K byte) boundary and use the result as *vaddr*.

The *ioctl* will fail if the virtual terminal is not in process mode or if the display is not in KD_GRAPHICS mode [EACCES]. It will fail if any of the arguments are out of range or not properly aligned [EFAULT]. It will fail if the display is already mapped to [EIO].

KDUNMAPDISP
: This call unmaps the direct access to the display and disables the direct usage of the I/O ports. The map must have been set up with KDMAPDISP by the current process. The memory where the display had been becomes zeroes. The *ioctl* will fail if the current process is not the one that did the mapping [EACCES].

VT_OPENQRY
: This call finds an available virtual terminal. The argument is a pointer to a long. The long will be filled with the number of the first available "VT" that no other process has open, or −1 if none are available.

VT_GETMODE
: This call determines what mode the active virtual terminal is currently in, either VT_AUTO or VT_PROCESS. The argument to the *ioctl* is the address of the following type of structure:

```
struct vt_mode {
  char   mode;      /* VT mode */
  char   waitv;     /* if set, hang on writes when not active */
  short  relsig;    /* signal to use for release request */
  short  acqsig;    /* signal to use for display acquired */
  short  frsig;     /* signal to use for forced release */
  }

#define VT_AUTO      0x00   /* automatic VT switching */
#define VT_PROCESS 0x01   /* process controls switching */
```

The **vt_mode** structure will be filled in with the current value for each field.

VT_SETMODE
: Set the virtual terminal mode. The argument is a pointer to a **vt_mode** structure, as defined above.

VT_RELDISP
: This call is used to tell the virtual terminal manager that the display has or has not been released by the process. A non-zero argument indicates that the display has been released; a zero argument indicates refusal to release the display.

VT_ACTIVATE
: This call makes the virtual terminal number specified in the argument the active VT. The VT manager will cause a switch to occur in the same manner as if a hotkey sequence had been typed at the keyboard. If the specified VT is not open or does not exist, the call will fail and *errno* will be set to ENXIO.

KIOCINFO
: This call tells the user what the device is.

GIO_SCRNMAP
: This call gets the screen mapping table from the kernel.

GIO_ATTR
: This call returns the current screen attribute. The bits are interpreted as follows:

  Bit 0 determines underlining for black and white monitors (1=underlining on).

  Bits 0-2, for color monitors only, select the foreground color. The following list indicates what colors are selected by the given value:

  The value 0 selects black.
  The value 1 selects red.
  The value 2 selects green.
  The value 4 selects blue.
  The value 5 selects magenta.
  The value 6 selects cyan.
  The value 7 selects white.

  Bit 3 is the intensity bit (1=blink on).

  Bits 4-6, for color monitors only, select the background color. For a list of colors and their values, see the list under foreground colors.

  Bit 7 is the blink bit (1=blink on).

GIO_COLOR
: This call returns a non-zero value if the current display is a color display; otherwise, it returns a zero.

PIO_SCRNMAP
: This call puts the screen mapping table in the kernel.

The screen mapping table maps extended ASCII (8-bit) characters to ROM characters. It is an array [256] of char (typedef **scrnmap_t**) and is indexed by extended ASCII values. The value of the elements of the array are the ROM character to display.

For example, the following program will cause the ASCII character # to be displayed as an English pound sign:

```
#include <sys/types.h>
#include <sys/at_ansi.h>
#include <sys/kd.h>
main() {
scrnmap_t scrntab;
        /* get screen mapping table of standard output */
if (ioctl(0,GIO_SCRNMAP, scrntab)==-1)
        {
        perror("screenmap read");
        exit(-1);
        }
/* 156 is the ROM value of English pound sign and 30
   is the ASCII value of '#'.
*/
scrntab[30] = 156;
if (ioctl(0, PIO_SCRNMAP, scrntab) == -1)
        {
        perror("screenmap write");
        exit(-1);
        }
}
```

FILES

/dev/console
/dev/vt00-*n*
/usr/include/sys/kd.h

SEE ALSO

stty(1), console(7), keyboard(7), termio(7).
ioctl(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

WARNINGS

Although it is possible to write character sequences which set arbitrary bits on the screen in any of the three graphics modes, this mode of operation is not currently supported.

It is currently not possible to access the 6845 start address registers. Thus, it is impossible to determine the beginning of the color monitor's screen memory.

The alternate/background color bit (bit 4) of the color select register does not appear to affect background colors in alphanumeric modes.

The low-resolution graphics mode appears to be 80 across by 100 down.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

fd − floppy disk (diskette) driver

DESCRIPTION

The diskette driver provides access to diskettes as both block and character devices. Diskettes must be formatted before they can be used (see *format*(1)). Both 512-byte and 1024-byte sectors with MFM encoding are supported. The driver controls up to two disk drives. The minor device number specifies both the drive number and the format of the diskette.

Diskette device file names (which correspond to a specific major and minor device) are in the following format:

/dev/{r}dsk/f#{dq}#{d}{t}

where **r** indicates a raw (character) interface to the disk, **f#** is the drive number, **d** or **q** indicates double or quad density (512 or 1024 byte sectors), **#** indicates the number of sectors per track, **d** indicates double-sided, and **t** indicates the entire disk (absence of this letter indicates that the first track of the diskette cannot be accessed). See the System Administration for New Users document in the *INTERACTIVE UNIX System Guide for New Users* for a discussion of XENIX diskette device files.

In order to minimize errors when using diskettes, the driver attempts to assure that the diskette is installed when needed, and that the operations requested have been completed before the device close is completed. In particular, the drive is checked for the presence of a diskette each time a read/write request is made to the drive. If this is not true (either because the diskette is not physically present or the door is open), the driver retries the request continually, at 5-second intervals. The message:

FD(*n*): diskette not present − please insert

appears after each attempt (the *n* represents the drive number). The INTR and QUIT signals are honored in this case, so that the process accessing the diskette drive in question will receive these signals (unless, of course, the process itself is ignoring them). In particular, if the diskette is removed prematurely or is not inserted soon enough, *no data is lost,* provided the correct diskette is inserted in the drive when the message to do so is displayed.

Ioctl Calls

V_GETPARMS This call is used to get information about the current drive configuration. The argument to the *ioctl* is the address of one of the following structures, defined in **<sys/vtoc.h>**, which will be filled in by the *ioctl*:

```
struct disk_parms {
   char      dp_type;           /* Disk type (see below) */
   unchar    dp_heads;          /* Number of heads */
   ushort    dp_cyls;           /* Number of cylinders */
   unchar    dp_sectors;        /* Number of sectors/track */
   ushort    dp_secsiz;         /* Number of bytes/sector */
                                /* for this partition: */
   ushort    dp_ptag;           /* Partition tag (not used) */
   ushort    dp_pflag;          /* Partition flag (not used) */
   ushort    dp_pstartsec;      /* Starting sector number */
   ushort    dp_pnumsec;        /* Number of sectors */
}

/* Disk types */
#define  DPT_WINI       1       /* Winchester disk */
#define  DPT_FLOPPY     2       /* Floppy */
#define  DPT_OTHER      3       /* Other type of disk */
#define  DPT_NOTDISK    0       /* Not a disk device */
```

For the floppy driver, the disk type will always be **DPT_FLOPPY**. The unused fields in the **disk_parms** structure are only applicable to fixed disks; however, returning the same structure from both the fixed disk driver and the diskette driver allows programs to be written that can understand either one.

V_FORMAT

This call is used to format tracks on a diskette. The argument passed to the *ioctl* is the address of one of the following structures, defined in **<sys/vtoc.h>**, containing the starting track, number of tracks, and interleave factor:

```
union io_arg {
   struct {
     ushort  start_trk;   /* first track */
     ushort  num_trks;    /* number of tracks to format */
     ushort  intlv;       /* interleave factor */
   } ia_fmt;
}
```

Formatting will start at the given track and will continue so that the given number of tracks are formatted, using the given interleave factor.

Note that the file descriptor must refer to the character (raw) special device for the desired drive, and the file must have been opened in exclusive mode (i.e., O_EXCL).

FILES

/dev/dsk/f0d9d, /dev/rdsk/f0d9d, ...
/dev/dsk/f0d9dt, /dev/rdsk/f0d9dt, ...
/dev/dsk/f0q15d, /dev/rdsk/f0q15d, ...
/dev/dsk/f0q15dt, /dev/rdsk/f0q15dt, ...
/dev/dsk/f0q9d, /dev/rdsk/f0q9d, ...
/dev/dsk/f0q9dt, /dev/rdsk/f0q9dt, ...
/dev/dsk/f0q18d, /dev/rdsk/f0q18d, ...
/dev/dsk/f0q18dt, /dev/rdsk/f0q18dt, ...

SEE ALSO

format(1M), mkpart(1M).
ioctl(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.
INTERACTIVE UNIX System Guide for New Users.

DIAGNOSTICS

The driver will retry failed transfers up to ten times. If the request still has not succeeded, the driver will display an appropriate message. Errors from the diskette controller, other than the above, are displayed as follows:

FD drv *n*, blk *b*: *drive error message*
FD controller *controller error message*

The first message occurs on an error after a transfer has begun, where *n* is the drive the error occurred on, and *b* is the block number that is being read or written. The *drive error message* is one of the messages appearing in the following list:

"Missing data address mark"
: The diskette may not be formatted properly.

"Cylinder marked bad"
: The accessed cylinder has been marked bad by the formatter.

"Seek error (wrong cylinder)"
: The drive positioned itself at the wrong cylinder when attempting to set up for the requested transfer.

"Uncorrectable data read error"
: A CRC error was detected when attempting to read the requested block from the drive.

"Sector marked bad"
: The accessed sector has been marked bad by the formatter.

"Missing header address mark"
: The diskette may not be formatted properly.

"Write protected"
: A write was attempted to a diskette that is currently write protected.

"Sector not found"
: The diskette may not be formatted properly.

"Data overrun"
: The system could not keep up with the requested transfer of data. (Should not occur.)

"Header read error"
: The diskette may not be formatted properly.

"Illegal sector specified"
: The driver is confused about the format of the diskette that has been inserted. (Should not occur.)

The second message occurs when there is a controller error during the setup for, or actual transfer of a block. The *controller error message* is one of the messages appearing in the following list:

"command timeout"
: The controller failed to complete the requested command in a reasonable length of time.

"status timeout"
: The controller failed to return its status after a command was completed.

"busy" During an attempt to access the controller, a timeout occurred.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

ft — floppy tape driver

DESCRIPTION

The floppy tape driver supports the Irwin and several QIC-40 floppy tape drives through a single character device file, **/dev/ftape**. It is not designed for direct reading or writing of the device file, but must be used with the *ftape*(1) user interface command (or other command written for use with this driver).

The floppy tape driver and the new floppy disk driver are two independent drivers sharing a common module that manages the floppy disk controller. The floppy disk driver is discussed separately in *fd*(7).

Note that when the floppy tape driver has been installed on the system, only one device, disk or tape, can be open at a time. This means that floppy disk to floppy tape, and floppy tape to floppy disk transfers are *not* possible.

The floppy tape driver supports the following drives:

Manufacturer	Model	Capacity
Irwin	225	20 MB
Irwin	245	40 MB
Irwin	285	80/120 MB
Alloy	APT-40	40 MB
CMS	Jumbo	40 MB
Wangtek	FAD3500	40 MB
Mountain	TD4000	40 MB

Floppy Tape Ioctl Calls

REWIND_ONLY
: Rewind to the beginning of the tape.

RETENSION
: Retension the tape.

ERASE
: Erase the tape.

SERVO
: Write servo information on the tape. The tape must be a new, unformatted tape or have been erased with a bulk tape eraser.

VERIFY
: Put the tape drive into a special verify mode that finds bad and marginally bad spots on the tape. It may be used in conjunction with bad blocking schemes to collect a list of known bad spots on the tape which may then be avoided. This *ioctl* command expects an integer parameter to turn the verify mode on or off. A value of 1 enables the verify mode and a value of 0 disables it. Once the verify mode is enabled it will stay enabled until the verify mode is explicitly turned off or until the device is closed.

QREAD, QWRITE, READWAIT, WRITEWAIT
: These *ioctl* commands provide the primary interface to the floppy tape driver. Different bad blocking and error correction schemes are currently used by various floppy tape drive vendors. The implementation of these schemes in not built into the driver, but is left to

application layer programs, e.g., *ftape*(1). The device driver provides an underlying base upon which practically any of these schemes may be implemented. The QREAD, QWRITE, READWAIT, and WRITEWAIT *ioctl* commands provide the interface by which read and write requests may be sent to the driver and by which the application program may determine when I/O operations on each of the requests has been completed. This allows multiple requests to be queued at the driver to facilitate streaming of the tape. The structure definitions and defines can be found in **<sys/mtioctl.h>**.

Each of these *ioctl* commands expects a structure of the form shown below:

```
/* structure for QREAD and QWRITE ioctls */
typedef struct {
    char *buffer;          /* pointer to actual data buffer */
    unsigned long badmap;/* kernel's idea of bad sectors after read */
                           /* or of bad sectors before write */
    unsigned long flags;
    long sec_num;          /* physical tape-block number */
}   IOBUFFER;
```

Data is read or written from the indicated address as specifed by the buffer field when the operation is performed. Floppy tapes are broken into tape blocks. Different tape formats have different numbers and sizes of tape blocks. The T_GETPARMS *ioctl* call may be used to determine the size and number of tape blocks on the cartridge tape currently installed in the tape drive. For the QREAD and QWRITE *ioctl* commands the *sec_num* field specifies a tape block which is to be read or written. The *badmap* field indicates known bad spots in the tape block for which the request is being made so that the driver can avoid them. The *flags* field is not currently used.

For the READWAIT and WRITEWAIT *ioctl* commands the *sec_num* field represents a tape block for which a read or write request has already been queued and for which the application program now wants to know the completion status. If the operation has been completed, the *ioctl* call will return immediately; if the operation has not been completed, the driver will wait for completion before returning. The *badmap* field provides a mechanism through which new bad spots on the tape are reported to the application program. If tape errors are encountered during the requested operation, bits in the *badmap* field are set to indicate the sectors that contained errors.

T_GETPARMS

This *ioctl* command returns information about the tape drive and the cartridge in use.

```
/* structure for T_GETPARMS calls */
struct tape_parms {
    int devtype;      /* type of drive */
    int sec_blk;      /* sectors per tape block */
    int numblocks;    /* total number of blocks on a tape */
    int tapetrks;     /* total number of tape tracks */
    int trkblks;      /* total number of blocks per tape track */
    int numbuffs;     /* total number of buffers available */
};
```

The *devtype* field contains the drive type and capacity encoded as follows:

Manufacturer	Model	Drive Capacity	Value of Bits 4-7	Value of Bits 1-2
Irwin	225	20 MB	1	2
Irwin	245	40 MB	1	1
Irwin	285	80/120 MB	1	0
Alloy	APT-40	40 MB	0	undefined
CMS	Jumbo	40 MB	2	undefined
Wangtek	FAD3500	40 MB	2	undefined
Mountain	TD4000	40 MB	2	undefined

The *sec_blk* field contains the number of sectors in one tape block on the drive currently installed.

The *numblocks* field contains the total number of tape blocks contained on the cartridge tape currently installed.

The *tapetrks* field contains the total number of tape tracks supported by the tape drive currently installed.

The *trkblks* field contains the total number of tape blocks contained in one tape track.

The driver contains internal buffers to assist in improving the performance of the tape subsystem and allowing the tape to stream whenever possible. The *numbuffs* field contains the number of internal buffers contained in the driver, and up to *numbuffs* QREAD or QWRITE requests may be queued in the driver at one time. If more than *numbuffs* QREAD or QWRITE requests are queued without an intervening READWAIT or WRITEWAIT, the driver with return an EIO error condition and not queue the request.

SEE ALSO

ftape(1).
ioctl(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

DIAGNOSTICS

The driver will retry failed transfers up to three times. If the request still has not succeeded, the driver will display an error message.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

hpdd – High Performance Disk Driver

DESCRIPTION

The *hpdd* module contains initialization procedures and low-level primitives for driver modules that run as part of the High Performance Disk Driver (HPDD) subsystem. Such drivers provide for improved system throughput, increased ease of maintenance, and quick implementation by making use of structures and code provided by the HPDD.

There are two types of modules associated with the HPDD: driver modules and low-level controller modules. Driver modules provide traditional UNIX System block and character device entry points and generate request queues in terms of HPDD-specific data elements and a generic set of commands common to most low-level hardware controllers. Currently, disk and tape drivers are supported by the *hpdd* subsystem (see *disk*(7) and *tape*(7)). It is anticipated that the list of supported drivers will grow fairly slowly. Low-level controller modules exist for each type of actual hardware interface supported by the HPDD. These determine whether boards configured are in fact present, translate the generic command set to actual hardware commands, and process hardware interrupts. A low-level module may be used by more than one driver (as is the case with a SCSI adapter used to communicate with both disk and tape drives) and may support more than one actual hardware controller. For example, the *athd* module is used to communicate with all AT-compatible fixed disk adapters (ST-506, RLL, or ESDI) that exist within a system.

Configuration

The file **/etc/conf/pack.d/gendev/space.c** contains a table called **gdev_init_routines**, which contains the names of the driver initialization routines for each HPDD driver module. The file **/etc/conf/sdevice.d/gendev** should indicate *all* interrupt levels to be handled by the HPDD low-level modules, as the control and scheduling of this activity is the responsibility of the HPDD module itself. Note that all devices supported by the HPDD must be at the same interrupt priority level. In addition, each driver will normally have a **space.c** file associated with it. Tables in these files describe the actual hardware expected to be present in a given system configuration. Note also that the names of the low-level controller modules configured must also be present in the modules list of the file **/etc/conf/cf.d/mdevice**.

SEE ALSO

disk(7), tape(7).

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

hub — Bell Technologies Hub6 serial card

DESCRIPTION

A Hub6 card provides six additional serial ports for IBM PC AT computers and compatibles. A single machine may contain up to four Hub6 cards, providing a total of 24 additional serial ports. Each line attached to a Hub6 serial card behaves as described in *asy*(7) and may be set to run at up to 9600 baud; see *tty*(7) for the appropriate encoding.

The Hub6 card uses the same 8250 communications and modem control chip used by IBM in the standard IBM Asynchronous Adaptor card. This part contains on-chip hardware for a large array of serial line and modem control functions, including fully automatic interrupt signalling.

FILES

/dev/tty4[a-x]	
/dev/tty4[A-X]	with modem controls
/etc/conf/pack.d/hub/space.c	

SEE ALSO

asy(7), termio(7), tty(7).
signal(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

icc – Intelligent Channel Controller (ICC) Multi-Port Serial Card

DESCRIPTION

The icc driver supports the Bell Technologies Intelligent Channel Controller (ICC) Multi-Port Serial Card for AT-bus systems. Up to 4 ICC cards are supported, for a total of up to 24 serial ports. Provision is made using the device minor number, as described below, to select an operation mode for a port of modem control, non-modem control, or output only. If a card for a port is not installed, an attempt to *open* it will fail.

If the port was opened with the modem control bit present in the minor device number (see below), modem control will be enabled. If enabled, the driver will wait in *open* until Data Carrier Detect (DCD) is present. Once opened, if DCD drops, the driver will return errors on any subsequent *read* or *write* of an asynchronous port by the user. If the port was opened as a controlling teletype, signal SIGHUP will be generated to the process which did the *open*.

Each port can be independently programmed for speed (50-38400 baud), character length, and parity. Output speed is always the same as input speed. The ports behave as described in *termio*(7). Setting B0 will cause a disconnect, setting EXTA will set 19200 baud, and setting EXTB will set 38400 baud.

Minor Device Numbers

The device minor number is the low-order 8 bits of the device number (see *mknod*(1M)). The low-order 5 bits (bits 0-4) of the minor device number are used to identify the asynchronous port number (0-23). Bit 7 enables modem control on the port. Thus, minor device 0 corresponds to the ICC board 0 serial port 0 with no modem control, while minor device 128 corresponds to the same port with modem control enabled. In a similar manner, bit 6 is used to indicate an output-only device.

A special case of the device minor number is the ICC control process that uses device number 255. Through this device number, the ICC control process downloads the board control program.

Ioctl Calls

The standard *tty* device *ioctl* calls are supported (see *termio*(7)). One exception is the use of XMT1EN in the *c_cflag* field which is redefined in **/usr/include/sys/iccioctl.h** as CTSRTS. Setting the CTSRTS indicator causes the port to utilize the “auto-enable” mode of operation in the SCC 8350/51 chip on the ICC board. In this mode, the UART is controlled by external signals to partially implement an RTS/CTS handshaking. The “auto-enable” mode simply stated means that the UART can transmit only if the CTS signal is true and can receive only when the DCD signal is true. For additional information on the operation in this mode, refer to the chip documentation.

Configuration

The ICC board provides for the selection of IRQ number, I/O space beginning address, and the starting address of the dual port RAM. Instructions for setting the board switches can be found in the User's Manual for the board. These switch settings for the board I/O address range, the dual-ported RAM, and the IRQ value must be communicated to the driver, which is done by modifying the files **/etc/conf/pack.d/icc/space.c** and **/etc/conf/sdevice.d/icc**. The **/etc/conf/pack.d/icc/config** file contains a definition for structure *icconf*, where field *port* contains the base I/O address for each ICC card defined by the board's dip switch setting, which is located on block S2 of the board itself, and field *sw* contains the hex value of the dip switch setting on the board located at block S1. The *sw* field specifies two characteristics of the board. Bits 0-2 of this field specify the ICC-to-system memory DMA channel selected, while bits 3-7 specify the base address for the ICC card's dual-ported memory. Board switch block S1 defines the dual-port RAM starting address for the 0.5 MB RAM. Switch S1 value should be at least 1 to establish a starting address above minimal motherboard 512KB RAM. Any changes to these dip switches on the ICC card must be reflected in the **/etc/conf/pack.d/icc/space.c** file or your driver will not work.

The file **/etc/conf/sdevice.d/icc** contains 10 entries. Each entry specifies an important specification regarding the ICC driver. The IRQ is specified in field 6. If the IRQ level for the board is changed, which is done by selecting 1 of 10 possible switch positions on the dip switch located on block S3 of the board. Field 6 of this file must reflect the corresponding IRQ level. The default IRQ level for ICC cards is 6. The I/O address ranges of the ICC cards are also specified in the file **/etc/conf/sdevice.d/icc**. The seventh and eighth entries in the **icc** file specify the beginning and ending addresses on the I/O bus through which the device communicates. These two fields specify the I/O address ranges for all cards (0-3). The default values for these two fields are *348* and *357*, respectively. For information on other fields in this file, refer to *sdevice*(4). Board switch block S1 defines the dual-port RAM starting address for the 0.5 MB RAM. Switch S1 value should be at least 1 to establish a starting address above minimal motherboard 512KB RAM. On some machines, especially those with cache controllers, it may be necessary to select a starting address that will place the board in a non-cached area of RAM. It is not necessary to set a different dual-port address for each card installed; however, each card must have a unique I/O address.

The shell script *mkicc* found at **/etc/icc** provides a convenient means of establishing the port path names. When executed, *mkicc* accepts as arguments the board numbers (blank separated) for which port paths are to be established.

FILES

/dev/ttyi[A-X]
: Serial port with modem control

/dev/ttyi[a-x]
: Serial port without modem control

/dev/ttyo[a-x]
: Serial output port without modem control

/dev/icc
: ICC control device

SEE ALSO

termio(7).
signal(2), sdevice(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.
The user's manual for your equipment.

BUGS

The VP/ix support provided allows text mode DOS applications and does not support the use of ICC ports as COM1 or COM2. Use of the CTSRTS mode does not fully implement a hardware CTSRTS handshaking.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

iolp – UnTerminal parallel printer interface

DESCRIPTION

The *iolp* driver is part of the UnTerminal I/O network adapter (IONA) facility. It provides the support for connecting a line printer to the parallel port of an UnTerminal remote station. The *close* waits until all output is competed before returning to the user. The lp driver allows only one process at a time to write to the printer. If it is already busy, an *open* for writing will return an error. However, the driver allows multiple *open*s to occur if they are *read-only*.

When the parallel printer interface is used on a remote station connected to an IONA board, the modem bit is disabled on the serial interface (see *iona*(7)) for that station. A limitation in the number of wires used by the UnTerminal hardware requires that the modem line be shared by both the asynchronous and the parallel printer interfaces.

FILES

/dev/lp*
/dev/iolp*

SEE ALSO

iona(7), lp(7), vna(7).

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

iona — UnTerminal I/O network adapter serial driver

DESCRIPTION

The *iona* driver supports up to eight serial ports on two UnTerminal I/O network adapter (IONA) boards. The ports can be programmed for speed (50-19200 baud), character length, and parity. Output speed is always the same as input speed. The ports behave as described in *termio*(7).

The asynchronous port is a character-at-a-time device for both input and output. This characteristic both limits the bandwidth which can be achieved over a line, and increases the interrupt loading on the central processor. In particular, file transfer programs such as *uucp*(1) may not function well at speeds over 9600 baud.

The baud rates of the serial adapter programmable baud-rate generator do not correspond exactly with system baud rates. In particular, setting B0 will cause a disconnect, setting EXTA will set 19200 baud, and setting EXTB will set 38400 baud. It is not possible to directly set 2000, 3600, or 7200 baud.

Because of a current hardware limitation, the modem control bit will be disabled when the parallel printer interface is used on an IONA remote station (see *iolp*(7)).

Minor Device Numbers

The asynchronous port is a character device. The low-order 4 bits (bits 0-3) of the minor device number correspond to the remote station at which the port is located. Bit 4 enables modem control on the port.

FILES

/dev/iona*

SEE ALSO

asy(7), iolp(7), termio(7), vna(7).
signal(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

kdmouse – built-in mouse device interface

DESCRIPTION

The *kdmouse* driver supports Micro Channel architecture mice and compatibles (such as the IBM PS/2 mouse) on machines with built-in mouse interfaces such as the COMPAQ 20e and the IBM PS/2 model 80. It allows applications to obtain information about the mouse's movements and the status of its buttons.

Programs are able to read directly from the device. The data returned corresponds to the byte sequences as defined in the *IBM PS/2 Technical Reference Manual*. Programs are not able to write to the *kdmouse* device.

Ioctl Call

MOUSEIOCREAD

This call returns status information about the mouse buttons and the mouse's relative movement since the previous MOUSEIOCREAD (or since mouse initialization, for the first MOUSEIOCREAD). The argument is the address of a *mouseinfo* structure; the following structure definition and **#defines** may be found in **<sys/mouse.h>**:

```
struct mouseinfo {
 unsigned char status;        /* see definitions below*/
 char xmotion, ymotion;       /* between -128 and 127*/
}
 #define BUT3STAT   1         /* button 3 status (1=down) */
 #define BUT2STAT   2         /* button 2 status (1=down) */
 #define BUT1STAT   4         /* button 1 status (1=down) */
 #define BUT3CHNG   8         /* button 3 changed? (1=yes) */
 #define BUT2CHNG   0x10      /* button 2 changed? (1=yes) */
 #define BUT1CHNG   0x20      /* button 1 changed? (1=yes) */
 #define MOVEMENT   0x40      /* mouse moved? (1=yes) */
```

FILES

/dev/kdmouse

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

keyboard — system console keyboard

DESCRIPTION

The system console is composed of two separate pieces: the keyboard and the display (see *display*(7)).

The keyboard is used to type data and to send certain control signals to the computer. UNIX System software performs terminal emulation on the console screen and keyboard, and, in doing so, makes use of several particular keys and key combinations. These keys and key combinations have special names that are unique to the UNIX System and may or may not correspond to the keytop labels on your keyboard.

When you press a key, one of the following happens:

- An ASCII value is entered.
- The meaning of another key, or keys, is changed.
- A string is sent to the computer.
- A function is initiated.

When a key is pressed (a keystroke), the keyboard sends a scancode to the computer. This scancode is interpreted by the keyboard driver. The actual code sequence delivered to the terminal input routine (see *termio*(7)) is defined by a set of internal tables in the driver. These tables can be modified by software (see the discussion of *ioctl* calls below). In addition, the driver can be instructed not to do translations, delivering the keyboard up/down scan codes directly.

Changing Meanings

The action performed by a key can be changed by using certain keys in combination. For example, **SHIFT** changes the ASCII values of the alphanumeric keys. Holding down **CTRL** while pressing another key sends a control code (such as **CTRL d**, **CTRL s**, and **CTRL q**). Holding down **ALT** also modifies a key's value. **SHIFT**, **CTRL**, and **ALT** can be used in combination.

Switching Screens

To switch the current screen, hold down **ALT SYSREQ** (also labelled **ALT PRINTSCRN** on some systems) and a key that identifies the desired screen. Any active screen may be selected by following **ALT SYSREQ** with F*n*, where F*n* is one of the function keys. F1 refers to the first virtual terminal screen, F2 refers to the second virtual terminal screen, etc. **ALT SYSREQ h** and **ALT SYSREQ** F8 refer to the main console display (**/dev/console**). The next active screen can be selected with **ALT SYSREQ n**, and the previous screen can be selected with **ALT SYSREQ p**.

The default screen switch enable sequence (**ALT SYSREQ**) is configurable. The **SYSREQ** table entry can be modified by software (see discussion of *ioctl* calls below).

Special Keys

The following table shows which keys on a typical console correspond to UNIX System keys. In this table, a hyphen (−) between keys means you must hold down the first key while pressing the second. The mapping between characters that generate signals and the signal actually generated is set with *stty*(1) and may be changed (see *stty*(1)).

Name	*Keytop*	*Action*
INTR	DEL	Stops current action and returns to the shell. This key is also called the RUB OUT or INTERRUPT key.
BACKSPACE	←	Deletes the first character to the left of the cursor. Note that the "cursor left" key also has a left arrow (←) on its keytop, but you cannot backspace using that key.
CTRL-d	CTRL-d	Signals the end of input from the keyboard; also exits current shell.
CTRL-h	CTRL-h	Deletes the first character to the left of the cursor. Also called the ERASE key.
CTRL-q	CTRL-q	Restarts printing after it has been stopped with CTRL-s.
CTRL-s	CTRL-s	Suspends printing on the screen (does not stop the program).
CTRL-u	CTRL-u	Deletes all characters on the current line. Also called the KILL key.
CTRL-\	CTRL-\	Quits current command and creates a **core** file, if allowed. (Recommended for debugging only.)
ESCAPE	ESC	Special code for some programs. For example, changes from insert mode to command mode in the *vi*(1) text editor.
RETURN	(down-left arrow or ENTER)	Terminates a command line and initiates an action from the shell.
F*n*	F*n*	Function key *n*. F1-F12 are unshifted, F13-F24 are shifted F1-F12, F25-F36 are CTRL-F1 through F12, and F37-F48 are CTRL-SHIFT-F1 through F12. The next F*n* keys (F49-F60) are on the number pad (unshifted): F49 – '7' F55 – '6' F50 – '8' F56 – '+' F51 – '9' F57 – '1' F52 – '–' F58 – '2' F53 – '4' F59 – '3' F54 – '5' F60 – '0'

Keyboard Map

The keyboard mapping structure is defined in **/usr/include/sys/kd.h**. Each key can have ten states. The first eight states are:

BASE
SHIFT
CTRL
ALT
CTRL-SHIFT
ALT-SHIFT
ALT-CTRL
ALT-CTRL-SHIFT

The two remaining states are indicated by two special bytes. The first byte is a "special state" byte whose bits indicate whether the key is "special" in one or more of the first eight states. The second byte is one of four codes represented by the characters C, N, B, or O which indicate how the lock keys affect the particular key.

The following table describes the default keyboard mapping. All values, except for special keywords (which are described later), are ASCII character values.

Heading	*Description*
SCAN CODE	This column contains the index to the table. Numbers 1 through 83 and 86 through 88 are actual scan codes generated by the keyboard hardware when a key is pressed. The numbers that do not have an entry in the KEY LOCATION column are not used. The remaining numbers are used for keys that generate a sequence of more than one code. There are no table entries for the scan code generated by releasing a key.
KEY LOCATION	This column contains the location number for the key on the 101-key keyboard map that precedes the table. Key location 45 (scan code 86) is for the international 102-key keyboard and is not shown (it is located between keys 44 and 46).
BASE	This column contains the normal value of a key press.
SHIFT	This column contains the value of a key press when the SHIFT key is also being held down.
LOCK	This column indicates which lock keys affect that particular key: – C indicates CAPS-LOCK – N indicates NUM-LOCK – B indicates both – O indicates locking is off

The remaining columns are the values of key presses when combinations of **CTRL**, **ALT**, and **SHIFT** are also held down.

The SRQTAB column entry is included in this table to provide a simple index of the default virtual terminal key selectors to the scan code to which it is assigned. The actual SRQTAB table is a stand-alone table which can be read or written via the KDGKBENT and KDSKBENT *ioctl* calls.

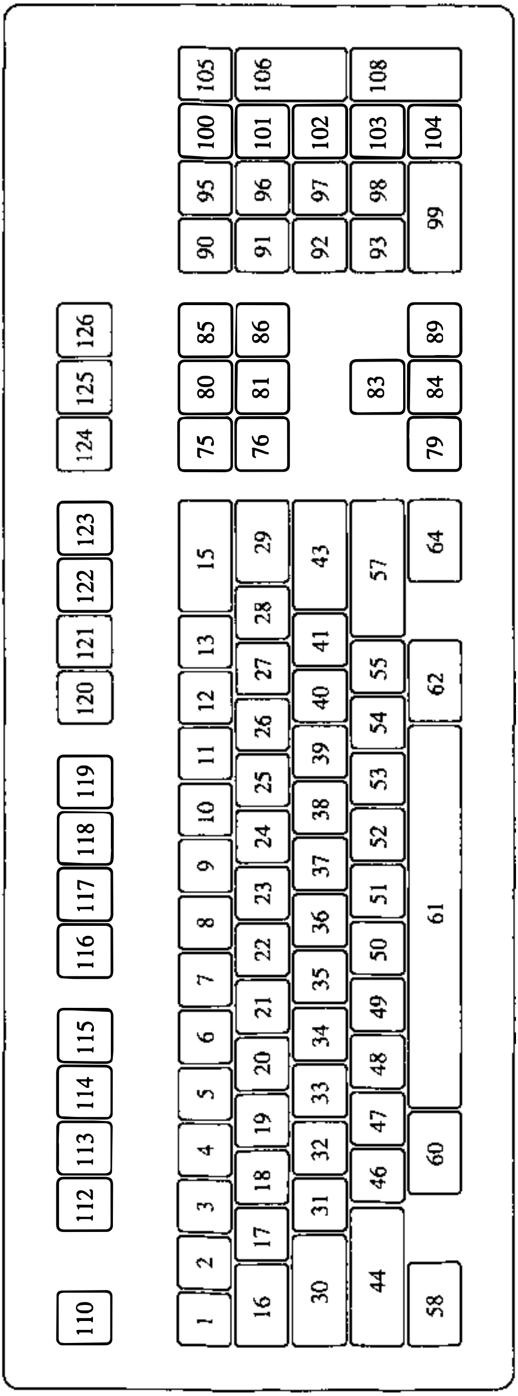


Figure 1. Assigned Location Numbers for Keys on a U.S. English (101-key) Keyboard

SCAN CODE	KEY LOCATION	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK	SRQTAB
0		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
1	110	esc	esc	esc	esc	esc	esc	esc	esc	O	nop
2	2	’1’	’!’	’1’	’1’	escn	escn	nop	nop	O	nop
3	3	’2’	’	’	’2’	nul	escn	escn	nop	nop	O
4	4	’3’	’#’	’3’	’3’	escn	escn	nop	nop	O	nop
5	5	’4’	’$’	’4’	’4’	escn	escn	nop	nop	O	nop
6	6	’5’	’%’	’5’	’5’	escn	escn	nop	nop	O	nop
7	7	’6’	’^’	’6’	rs	escn	escn	nop	nop	O	nop
8	8	’7’	’&’	’7’	’7’	escn	escn	nop	nop	O	nop
9	9	’8’	’*’	’8’	’8’	escn	escn	nop	nop	O	nop
10	10	’9’	’(’	’9’	’9’	escn	escn	nop	nop	O	nop
11	11	’0’	’)’	’0’	’0’	escn	escn	nop	nop	O	nop
12	12	’-’	’_’	’-’	ns	escn	escn	nop	nop	O	nop
13	13	’=’	’+’	’=’	’=’	escn	escn	nop	nop	O	nop
14	15	bs	bs	bs	bs	bs	bs	bs	bs	O	nop
15	16	ht	btab	ht	btab	ht	btab	ht	btab	O	nop
16	17	’q’	’Q’	dc1	dc1	escn	escn	nop	nop	C	nop
17	18	’w’	’W’	etb	etb	escn	escn	nop	nop	C	nop
18	19	’e’	’E’	enq	enq	escn	escn	nop	nop	C	nop
19	20	’r’	’R’	dc2	dc2	escn	escn	nop	nop	C	nop
20	21	’t’	’T’	dc4	dc4	escn	escn	nop	nop	C	nop
21	22	’y’	’Y’	em	em	escn	escn	nop	nop	C	nop
22	23	’u’	’U’	nak	nak	escn	escn	nop	nop	C	nop
23	24	’i’	’I’	ht	ht	escn	escn	nop	nop	C	nop
24	25	’o’	’O’	si	si	escn	escn	nop	nop	C	nop
25	26	’p’	’P’	dle	dle	escn	escn	nop	nop	C	PREV
26	27	’[’	’{’	esc	nop	escn	escn	nop	nop	O	nop
27	28	’]’	’}’	gs	nop	escn	escn	nop	nop	O	nop
28	43	cr	cr	cr	cr	cr	cr	cr	cr	O	nop
29	58	lctrl	lctrl	lctrl	lctrl	lctrl	lctrl	lctrl	lctrl	O	nop
30	31	’a’	’A’	soh	soh	escn	escn	nop	nop	C	nop
31	32	’s’	’S’	dc3	dc3	escn	escn	nop	nop	C	nop
32	33	’d’	’D’	eot	eot	escn	escn	nop	nop	C	nop

SCAN CODE	KEY LOCATION	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK	SRQTAB
33	34	'f'	'F'	ack	ack	escn	escn	nop	nop	C	FNEXT
34	35	'g'	'G'	bel	bel	escn	escn	nop	nop	C	nop
35	36	'h'	'H'	bs	bs	escn	escn	nop	nop	C	VT1
36	37	'j'	'J'	nl	nl	escn	escn	nop	nop	C	nop
37	38	'k'	'K'	vt	vt	escn	escn	nop	nop	C	nop
38	39	'l'	'L'	np	np	escn	escn	nop	nop	C	nop
39	40	';'	':'	';'	':'	escn	escn	nop	nop	O	nop
40	41	'''	'"'	'''	'"'	escn	escn	nop	nop	O	nop
41	1	'`'	'~'	'`'	'~'	escn	escn	nop	nop	O	nop
42	44	lshift	lshift	lshift	lshift	lshift	lshift	lshift	lshift	O	nop
43	29	'\\'	'|'	fs	'|'	escn	escn	nop	nop	O	nop
44	46	'z'	'Z'	sub	sub	escn	escn	nop	nop	C	nop
45	47	'x'	'X'	can	can	escn	escn	nop	nop	C	nop
46	48	'c'	'C'	etx	etx	escn	escn	nop	nop	C	nop
47	49	'v'	'V'	syn	syn	escn	escn	nop	nop	C	nop
48	50	'b'	'B'	stx	stx	escn	escn	nop	nop	C	nop
49	51	'n'	'N'	so	so	escn	escn	nop	nop	C	NEXT
50	52	'm'	'M'	cr	cr	escn	escn	nop	nop	C	nop
51	53	','	'<'	','	'<'	escn	escn	nop	nop	O	nop
52	54	'.'	'>'	'.'	'>'	escn	escn	nop	nop	O	nop
53	55	'/'	'?'	'/'	ns	escn	escn	nop	nop	O	nop
54	57	rshift	rshift	rshift	rshift	rshift	rshift	rshift	rshift	O	nop
55	100	'*'	'*'	'*'	'*'	escn	escn	nop	nop	O	nop
56	60	lalt	lalt	lalt	lalt	lalt	lalt	lalt	lalt	O	nop
57	61	' '	' '	nul	nul	escn	escn	nop	nop	O	nop
58	30	clock	clock	clock	clock	clock	clock	clock	clock	O	nop
59	112	fkey1	fkey13	fkey25	fkey37	fkey1	fkey13	fkey25	fkey37	O	VT2
60	113	fkey2	fkey14	fkey26	fkey38	fkey2	fkey14	fkey26	fkey38	O	VT3
61	114	fkey3	fkey15	fkey27	fkey39	fkey3	fkey15	fkey27	fkey39	O	VT4
62	115	fkey4	fkey16	fkey28	fkey40	fkey4	fkey16	fkey28	fkey40	O	VT5
63	116	fkey5	fkey17	fkey29	fkey41	fkey5	fkey17	fkey29	fkey41	O	VT6
64	117	fkey6	fkey18	fkey30	fkey42	fkey6	fkey18	fkey30	fkey42	O	VT7

SCAN CODE	KEY LOCATION	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK	SRQTAB
65	118	fkey7	fkey19	fkey31	fkey43	fkey7	fkey19	fkey31	fkey43	O	VT8
66	119	fkey8	fkey20	fkey32	fkey44	fkey8	fkey20	fkey32	fkey44	O	VT1
67	120	fkey9	fkey21	fkey33	fkey45	fkey9	fkey21	fkey33	fkey45	O	nop
68	121	fkey10	fkey22	fkey34	fkey46	fkey10	fkey22	fkey34	fkey46	O	nop
69	90	nlock	nlock	nlock	nlock	nlock	nlock	nlock	nlock	O	nop
70	125	slock	slock	brk	brk	slock	slock	brk	brk	O	nop
71	91	fkey49	'7'	fkey49	'7'	fkey49	escn	nop	nop	N	nop
72	96	fkey50	'8'	fkey50	'8'	fkey50	escn	nop	nop	N	nop
73	101	fkey51	'9'	fkey51	'9'	fkey51	escn	nop	nop	N	nop
74	105	fkey52	'-'	fkey52	'-'	fkey52	escn	nop	nop	N	nop
75	92	fkey53	'4'	fkey53	'4'	fkey53	escn	nop	nop	N	nop
76	97	fkey54	'5'	fkey54	'5'	fkey54	escn	nop	nop	N	nop
77	102	fkey55	'6'	fkey55	'6'	fkey55	escn	nop	nop	N	nop
78	106	fkey56	'+'	fkey56	'+'	fkey56	escn	nop	nop	N	nop
79	93	fkey57	'1'	fkey57	'1'	fkey57	escn	nop	nop	N	nop
80	98	fkey58	'2'	fkey58	'2'	fkey58	escn	nop	nop	N	nop
81	103	fkey59	'3'	fkey59	'3'	fkey59	escn	nop	nop	N	nop
82	99	fkey60	'0'	fkey60	'0'	fkey60	escn	nop	nop	N	nop
83	104	del	'.'	del	'.'	del	escn	rboot	nop	N	nop
84	124	fkey60	fkey26	fkey60	nop	sysreq	sysreq	sysreq	sysreq	O	nop
85	84	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	O	nop
*86	45	'<'	'>'	'<'	'>'	escn	escn	nop	nop	O	nop
87	122	fkey11	fkey23	fkey35	fkey47	fkey11	fkey23	fkey35	fkey47	O	nop
88	123	fkey12	fkey24	fkey36	fkey48	fkey12	fkey24	fkey36	fkey48	O	nop
89		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
90		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
91		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
92		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
93		nop	nop	nop	nop	nop	nop	nop	nop	O	nop

*Applicable only for 102-key keyboard.

SCAN CODE	KEY LOCATION	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK	SRQTAB
94		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
95		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
96		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
97		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
98		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
99		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
100		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
101		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
102		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
103		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
104		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
105		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
106		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
107		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
108		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
109		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
110		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
111	85	fkey51	fkey51	nop	nop	nop	nop	nop	nop	O	nop
112		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
113		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
114	62	ralt	ralt	ralt	ralt	ralt	ralt	ralt	ralt	O	nop
115	64	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	O	nop
116	108	cr	cr	cr	cr	cr	cr	cr	cr	O	nop
117	95	'/'	'/'	nop	nop	escn	escn	nop	nop	O	nop
118		nop	nop	nop	nop	nop	nop	nop	nop	O	nop
119	126	brk	brk	brk	brk	brk	brk	brk	brk	O	nop
120	83	fkey50	fkey50	nop	nop	nop	nop	nop	nop	O	nop
121	76	del	del	del	del	del	del	del	del	O	nop
122	81	fkey57	fkey57	nop	nop	nop	nop	nop	nop	O	nop
123	75	fkey60	fkey60	nop	nop	nop	nop	nop	nop	O	nop
124	79	fkey53	fkey53	nop	nop	nop	nop	nop	nop	O	nop
125	89	fkey55	fkey55	nop	nop	nop	nop	nop	nop	O	nop
126	86	fkey59	fkey59	nop	nop	nop	nop	nop	nop	O	nop
127	80	fkey49	fkey49	nop	nop	nop	nop	nop	nop	O	nop

The following table lists the value of each of the special keywords used in the preceding tables. The keywords are only used in the preceding tables for readability. In the actual keyboard map, a special keyword is represented by its value with the corresponding "special state" bit being set.

Name	*Value*	*Meaning*
nop	0	No operation – no action from keypress
lshift	2	Left-hand shift
rshift	3	Right-hand shift
clock	4	Caps lock
nlock	5	Numeric lock
slock	6	Scroll lock
alt	7	Alt key
btab	8	Back tab key – generates fixed sequence (ESC[z)
ctrl	9	Control key
lalt	10	Left-hand alt key
ralt	11	Right-hand alt key
lctrl	12	Left-hand control key
rctrl	13	Right-hand control key
fkey1	27	Function key #1
.		.
.		.
.		.
fkey96	122	Function key #96
sysreq	123	System request
brk	124	Break key
escn	125	Generate an ESC N *x* sequence, where *x* is the un-alt'ed value of the scan code
rboot	128	Reboot system
debug	129	Invoke kernel debugger
NEXT	130	Switch to next virtual terminal on queue
PREV	131	Switch to previous virtual terminal on queue
FNEXT	132	Forced switch to next virtual terminal on queue
VT1	134	Switch to virtual terminal #1
.		.
.		.
.		.
VT8	141	Switch to virtual terminal #8

The following table lists names and decimal values for ASCII characters in the preceding table. Names are used in place of numeric constants to make it easier to read the scan code table. Only the decimal values are placed in the *ioctl* buffer. These values are taken from *ascii*(5).

Name	*Value*	*Name*	*Value*
nul	0	dc1	17
soh	1	dc2	18
stx	2	dc3	19
etx	3	dc4	20
eot	4	nak	21
enq	5	syn	22
ack	6	etb	23
bel	7	can	24
bs	8	em	25
ht	9	sub	26
nl	10	esc	27
vt	11	fs	28
np	12	gs	29
cr	13	rs	30
so	14	ns	31
si	15	del	127
dle	16		

String Key Mapping

The string mapping table is an array of 512 bytes (typedef **strmap_t**) containing null-terminated strings that redefine the function keys. The first null-terminated string is assigned to the first function key, the second string is assigned to the second function key, etc.

There is no limit to the length of any particular string as long as the whole table does not exceed 512 bytes, including nulls. To make a string a null, add extra null characters. The following table contains default function key values.

Default Function Key Values			
Function Key #	*Function*	*Function Key #*	*Function*
1	ESC OP	32	ESC OW
2	ESC OQ	33	ESC OX
3	ESC OR	34	ESC OY
4	ESC OS	35	ESC OZ
5	ESC OT	36	ESC OA
6	ESC OU	37	ESC Op
7	ESC OV	38	ESC Oq
8	ESC OW	39	ESC Or
9	ESC OX	40	ESC Os
10	ESC OY	41	ESC Ot
11	ESC OZ	42	ESC Ou
12	ESC OA	43	ESC Ov
13	ESC Op	44	ESC Ow
14	ESC Oq	45	ESC Ox
15	ESC Or	46	ESC Oy
16	ESC Os	47	ESC Oz
17	ESC Ot	48	ESC Oa
18	ESC Ou	49	ESC [H
19	ESC Ov	50	ESC [A
20	ESC Ow	51	ESC [V
21	ESC Ox	52	ESC [S
22	ESC Oy	53	ESC [D
23	ESC Oz	54	ESC [G
24	ESC Oa	55	ESC [C
25	ESC OP	56	ESC [T
26	ESC OQ	57	ESC [Y
27	ESC OR	58	ESC [B
28	ESC OS	59	ESC [U
29	ESC OT	60	ESC [@
30	ESC OU	61	ESC [2
31	ESC OV		

Ioctl Calls

KDGKBMODE

This call gets the current keyboard mode. It returns one of the following values, as defined in **/usr/include/sys/kd.h**:

```
#define   K_RAW     0x00   /* Send raw scan codes */
#define   K_XLATE   0x01   /* Translate to ASCII */
```

KDSKBMODE

This call sets the keyboard mode. The argument to the call is either **K_RAW** or **K_XLATE.** By using raw mode, the program can see the raw up/down scan codes from the keyboard. In translate mode, the translation tables are used to generate the appropriate character code.

KDGKBTYPE

This call gets the keyboard type. It returns one of the following values, as defined in **/usr/include/sys/kd.h**:

```
#define   KB_84      0x00   /*84 key keyboard*/
#define   KB_101     0x01   /*101 key keyboard*/
#define   KB_OTHER   0x03   /*Other type keyboard*/
```

KDGKBENT

This call reads one of the entries in the translation tables. The argument to the call is the address of one of the following structures, defined in **/usr/include/sys/kd.h**, with the first two fields filled in:

```
struct kbentry {
  unchar    kb_table;     /* Table to use */
  unchar    kb_index;     /* Entry in table */
  ushort    kb_value;     /* Value to get/set */
};
```

Valid values for the **kb_table** field are:

```
#define  K_NORMTAB       0x00   /* Base */
#define  K_SHIFTTAB      0x01   /* Shifted */
#define  K_ALTTAB        0x02   /* Alt */
#define  K_ALTSHIFTTAB   0x03   /* Shifted alt */
#define  K_SRQTAB        0x04   /* Sysreq table */
```

The *ioctl* will get the indicated entry from the indicated table and return it in the **kb_value** field.

The **kb_value** field is 16 bits, with flags in the high-order 8 bits and the character code in the low-order 8 bits. The values that can be set in the flag byte, as defined in **/usr/include/sys/kd.h**, are as follows:

```
/* Flag bits */
#define   NUMLCK     0x8000   /* key is affected by num lock */
#define   CAPLCK     0x4000   /* key is affected by caps lock */
#define   CTLKEY     0x2000   /* key is affected by control key */

/* Key types */
#define   NORMKEY    0x0000   /* key is a normal key */
#define   SHIFTKEY   0x0100   /* key is a shift key */
#define   BREAKKEY   0x0200   /* key is a break key */
#define   SS2PFX     0x0300   /* prefix key with <ESC> N */
#define   SS3PFX     0x0400   /* prefix key with <ESC> O */
#define   CSIPFX     0x0500   /* prefix key with <ESC> [ */
#define   NOKEY      0x0f00   /* key sends nothing */
```

KDSKBENT

This call sets an entry in one of the translation tables. It uses the same structure as the KDGKBENT *ioctl*, but with the third field filled in with the value that should be placed in the translation table. This can be used to partially or completely remap the keyboard.

The *kd* driver provides support for virtual terminals. Two *ioctl*s are provided for virtual terminal support:

VT_GETSTATE

The **VT_GETSTATE** *ioctl* returns global virtual terminal state information. It returns the active virtual terminal in the

v_active field, and the number of active virtual terminals and a bit mask of the global state in the **vt_state** field, where "bit x" is the state of "vt x" (1 indicates that the virtual terminal is open).

VT_SENDSIG
: The **VT_SENDSIG** *ioctl* specifies a signal (in **vt_signal**) to be sent to a bit mask of virtual terminals (in **vt_state**).

The data structure used by the **VT_GETSTATE** and **VT_SENDSIG** *ioctl*s is:

```
struct vt_stat {
    ushort v_active; /* active vt */
    ushort v_signal; /* signal to send (VT_SENDSIG) */
    ushort v_state;  /* vt bit mask (VT_SENDSIG and VT_GETSTATE) */
};
```

and is defined in **/usr/include/sys/vt.h**.

VT_OPENQRY
: The **VT_OPENQRY** *ioctl* is used to get the next available virtual terminal. This value is set in the last argument of the *ioctl*(2) call.

GIO_KEYMAP
: This call gets the entire keyboard mapping table from the kernel. The structure of the argument is given in **/usr/include/sys/kd.h**.

PIO_KEYMAP
: This call sets the entire keyboard mapping table. The structure of the argument is given in **/usr/include/sys/kd.h**.

GIO_STRMAP
: This call gets the string key mapping table from the kernel. The structure of the argument is given in **/usr/include/sys/kd.h**.

PIO_STRMAP
: This call sets the string key mapping table. The structure of the argument is given in **/usr/include/sys/kd.h**.

TIOCKBOF
: Extended character codes are disabled. This is the default mode.

TIOCKBON
: Allows extended characters to be transmitted to the user program. The extended characters are transmitted as a null byte followed by a second byte containing the character's extended code. When a true null byte is sent, it is transmitted as two consecutive null bytes.

KDSETRAD
: This call allows the application to set the keyboard typematic rate and delay. The default value is *0x20*, which sets the keyboard at 30 characters per second with a ½-second start-up delay. For other allowable values, refer to the *IBM Technical Reference Manual*.

When the keyboard is fully enabled, an 8-bit character code can be obtained by holding down the **ALT** key and entering the 3-digit decimal value of the character from the numeric keypad. The character is transmitted when the **ALT** key is released.

Some keyboard characters have special meaning. Under default operations, pressing the **DELETE** key generates an interrupt signal which is sent to all processes designated with the associated control terminal. When the keyboard is fully enabled, holding down the **ALT** key and pressing the 8 key on the home keyboard (not on the numeric keypad) returns a null byte followed by 0x7F. This will produce the same effect as the **DELETE** key (0x7F) unless you have executed the *stty*(1) command with the **−isig** option.

KBENABLED
: If the keyboard is fully enabled (TIOCKBON), a nonzero value will be returned. If the keyboard is not fully enabled (TIOCKBOF), a value of zero will be returned.

GETFKEY
: Obtains the current definition of a function key. The argument to the call is the address of one of the following structures defined in **/usr/include/sys/kd.h**:

```
struct fkeyarg {
   unsigned int keynum;
   char         keydef [MAXFK];   /*Comes from ioctl.h via comcrt.h*/
   char         flen;
};
```

The function key number must be passed in *keynum* (see *arg* structure above). The string currently assigned to the key will be returned in *keydef*, and the length of the string will be returned in *flen* when the *ioctl* is performed.

SETFKEY
: Assigns a given string to a function key. It uses the same structure as the GETFKEY *ioctl*. The function key number must be passed in *keynum*, the string must be passed in *keydef*, and the length of the string (number of characters) must be passed in *flen*.

FILES

/dev/console
/dev/vt00-n
/usr/include/sys/kd.h

SEE ALSO

stty(1), console(7), display(7), termio(7),
ioctl(2), ascii(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

log − interface to STREAMS error logging and event tracing

DESCRIPTION

log is a STREAMS software device driver that provides an interface for the STREAMS error logging and event tracing processes [*strerr*(1M), *strace*(1M)]. *log* presents two separate interfaces: a function call interface in the kernel through which STREAMS drivers and modules submit *log* messages; and a subset of *ioctl*(2) system calls and STREAMS messages for interaction with a user level error logger, a trace logger, or processes that need to submit their own *log* messages.

Kernel Interface

log messages are generated within the kernel by calls to the function *strlog*:

```
strlog(mid, sid, level, flags, fmt, arg1, ...)
short mid, sid;
char level;
ushort flags;
char *fmt;
unsigned arg1;
```

Required definitions are contained in **<sys/strlog.h>** and **<sys/log.h>**. *mid* is the STREAMS module id number for the module or driver submitting the *log* message. *sid* is an internal sub-id number usually used to identify a particular minor device of a driver. *level* is a tracing level that allows for selective screening out of low priority messages from the tracer. *flags* are any combination of SL_ERROR (the message is for the error logger), SL_TRACE (the message is for the tracer), SL_FATAL (advisory notification of a fatal error), and SL_NOTIFY (request that a copy of the message be mailed to the system administrator). *fmt* is a *printf(3S)* style format string, except that %s, %e, %E, %g, and %G conversion specifications are not handled. Up to NLOGARGS (currently 3) numeric or character arguments can be provided.

User Interface

log is opened via the clone interface, **/dev/log**. Each open of **/dev/log** obtains a separate *stream* to *log*. In order to receive *log* messages, a process must first notify *log* whether it is an error logger or trace logger via a STREAMS I_STR *ioctl* call (see below). For the error logger, the I_STR *ioctl* has an ic_cmd field of I_ERRLOG with no accompanying data. For the trace logger, the *ioctl* has an ic_cmd field of I_TRCLOG, and must be accompanied by a data buffer containing an array of one or more struct trace_ids elements. Each trace_ids structure specifies an *mid*, *sid*, and *level* from which message will be accepted. *strlog* will accept messages whose *mid* and *sid* exactly match those in the trace_ids structure, and whose level is less than or equal to the level given in the trace_ids structure. A value of −1 in any of the fields of the trace_ids structure indicates that any value is accepted for that field.

At most one trace logger and one error logger can be active at a time. Once the logger process has identified itself via the *ioctl* call, *log* will begin sending up messages subject to the restrictions noted above.

These messages are obtained via the *getmsg(2)* system call. The control part of this message contains a log_ctl structure, which specifies the *mid*, *sid*, *level*, *flags*, time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, and a sequence number. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of *log* messages can be determined.

Different sequence numbers are maintained for the error and trace logging *streams*, and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic, some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by NLOGARGS words for the arguments to the format string, aligned on the first word boundary following the format string.

A process may also send a message of the same structure to *log*, even if it is not an error or trace logger. The only fields of the log_ctl structure in the control part of the message that are accepted are the level and flags fields; all other fields are filled in by *log* before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to NLOGARGS) must be packed one word each, on the next word boundary following the end of the format string.

Attempting to issue an I_TRCLOG or I_ERRLOG when a logging process of the given type already exists will result in the error ENXIO being returned. Similarly, ENXIO is returned for I_TRCLOG *ioctls* without any trace_ids structures, or for any unrecognized I_STR *ioctl* calls. Incorrectly formatted *log* messages sent to the driver by a user process are silently ignored (no error results).

EXAMPLES

Example of I_ERRLOG notification.

```
struct strioctl ioc;

ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0;          /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;

ioctl(log, I_STR, &ioc);
```

Example of I_TRCLOG notification.

```
struct trace_ids tid[2];

tid[0].ti_mid = 2;
tid[0].ti_sid = 0;
tid[0].ti_level = 1;

tid[1].ti_mid = 1002;
tid[1].ti_sid = -1;         /* any sub-id will be allowed */
tid[1].ti_level = -1;       /* any level will be allowed */
```

```
ioc.ic_cmd = I_TRCLOG;
ioc.ic_timout = 0;
ioc.ic_len = 2 * sizeof(struct trace_ids);
ioc.ic_dp = (char *)tid;

ioctl(log, I_STR, &ioc);
```

Example of submitting a *log* message (no arguments).

```
struct strbuf ctl, dat;
struct log_ctl lc;
char *message = "Don't forget to pick up some milk on the way home";

ctl.len = ctl.maxlen = sizeof(lc);
ctl.buf = (char *)&lc;

dat.len = dat.maxlen = strlen(message);
dat.buf = message;

lc.level = 0;
lc.flags = SL_ERROR|SL_NOTIFY;

putmsg(log, &ctl, &dat, 0);
```

FILES

/dev/log, <sys/log.h>, <sys/strlog.h>

SEE ALSO

strace(1M), strerr(1M), clone(7).

intro(2), getmsg(2), putmsg(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

STREAMS Programmer's Guide.

NAME

lp – parallel printer interface

DESCRIPTION

The *lp* driver supports both the primary printer adapter (on the monochrome display adapter card) and secondary parallel printer adapters simultaneously. Up to three printers are supported. If an adapter for a printer is not installed, an attempt to *open* it will fail. The *close* waits until all output is completed before returning to the user. The *lp* driver allows only one process at a time to write to the adapter. If it is already busy, an *open* for writing will return an error. However, the driver allows multiple *open*s to occur if they are *read-only*.

The parallel printer adapters are character devices. The minor device number corresponds to the primary or secondary parallel printer adapter. Thus, minor device 0 corresponds to the primary parallel printer adapter, while minor device 1 corresponds to the secondary adapter.

The line printer driver allows users to determine how often the system will check whether a printer job is done and how often the system will send a message to the console alerting the operator that a printer needs attention. The two tunable parameters are:

Parameter	*Default Value*
LP_POLLINT	(HZ/20)
LP_WARNINT	(HZ*120)

where the system parameter HZ (hertz) is the number of ticks per second of the system clock (100).

This enhancement improves performance when using certain monochrome adapter ports. Some parallel port interfaces, such as the parallel port on many monochrome display adapters, do not latch (retain) their interrupt signals. This results in a loss of expected completion (READY) interrupts. To prevent this loss from slowing or stopping printer output, common code shared by the *lp* and *slp* drivers now uses a timeout mechanism to poll the interfaces for completion. **LP_POLLINT** (default 1/20th of a second) defines this polling interval.

When the printer driver detects a condition requiring operator intervention (such as paper-out), it writes a message on the console. **LP_WARNIT** (default 2 minutes) defines (in minutes) the interval between these warnings.

These parameters may be added to your system using *kconfig*. Select the option ADD TUNABLE PARAMETERS on the CONFIGURE KERNEL menu. The system will prompt you for the parameter name and its value.

FILES

/dev/lp*

SEE ALSO

kconfig(1).
"INTERACTIVE UNIX Operating System Maintenance Procedures" in the *INTERACTIVE UNIX Operating System Guide*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

mcesdi – low-level controller module

DESCRIPTION

The *mcesdi* module provides low-level interface routines between the High Performance Disk Driver (see *hpdd*(7)) and the controllers using an IBM/ESDI interface in a Micro Channel-compatible machine. The driver can be configured for disk support for one or two controller boards if the machine supports a multiple ESDI controller configuration. However, it can also be configured to run with other non-ESDI controller boards. The low-level code determines if the controller is, in fact, present at the configured address (see *disk*(7) and *tape*(7))) and what devices are attached to it.

For information on configuring a kernel to include the *mcesdi* module, refer to the "INTERACTIVE UNIX Operating System Maintenance Procedures."

Board Configuration

The reference diskette that accompanies the Micro Channel machine should be used for standard configurations of the machine. If a machine is configured with an IBM/ESDI and an IBM/ST506 (see *mcst*(7)) interface controller, both controllers must share the same interrupt level.

SEE ALSO

disk(7), hpdd(7), mcst(7), tape(7),
"INTERACTIVE UNIX Operating System Maintenance Procedures" in the *INTERACTIVE UNIX Operating System Guide* .

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

mcst – low-level controller module

DESCRIPTION

The *mcst* module provides low-level interface routines between the High Performance Disk Driver (see *hpdd*(7)) and controllers using IBM/ST506 interface in a Micro Channel-compatible machine. The driver can be configured for disk support for one or two controller boards if the machine supports a multiple ST506 controller configuration. However, it can be configured to run with other non-ST506 controller boards. The low-level code determines if the controller is, in fact, present at the configured address (see *disk*(7) and *tape*(7)) and what devices are attached to it.

For information on configuring a kernel to include the *mcst* module, refer to the "INTERACTIVE UNIX Operating System Maintenance Procedures."

Board Configuration

The reference diskette that accompanies the Micro Channel machine should be used for standard configurations of the machine. If a machine is configured with an IBM/ST506 and an IBM/ESDI (see *mcesdi*(7)) interface controller, both controllers must share the same interrupt level.

SEE ALSO

disk(7), hpdd(7), mcesdi(7), tape(7).
"INTERACTIVE UNIX Operating System Maintenance Procedures" in the *INTERACTIVE UNIX Operating System Guide*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

mem, kmem − core memory

DESCRIPTION

The file **/dev/mem** is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system.

Byte addresses in **/dev/mem** are interpreted as memory addresses. References to nonexistent 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 **/dev/kmem** is the same as */dev/mem* except that kernel virtual memory rather than physical memory is accessed.

I/O is not memory mapped, and the per-process data begins at virtual address 0xE0000000.

FILES

/dev/mem
/dev/kmem

WARNING

Some of **/dev/kmem** cannot be read because of write-only addresses or unequipped memory addresses.

NAME

mouse – mouse device interface

DESCRIPTION

The *mouse* driver supports the Microsoft Inport Bus Mouse on the AT release and the IBM PS/2 mouse on the PS/2 release. It allows applications to obtain information about the mouse's movements and the status of its buttons.

Ioctl Call

MOUSEIOCREAD

This call returns status information about the mouse buttons and the mouse's relative movement since the previous MOUSEIOCREAD (or since mouse initialization, for the first MOUSEIOCREAD). The argument is the address of a *mouseinfo* structure; the following structure definition and **#defines** may be found in **<sys/mouse.h>**:

```
struct mouseinfo {
 unsigned char status;          /* see definitions below*/
 char xmotion, ymotion;         /* between -128 and 127*/
}
 #define BUT3STAT   1           /* button 3 status (1=down) */
 #define BUT2STAT   2           /* button 2 status (1=down) */
 #define BUT1STAT   4           /* button 1 status (1=down) */
 #define BUT3CHNG   8           /* button 3 changed? (1=yes) */
 #define BUT2CHNG   0x10        /* button 2 changed? (1=yes) */
 #define BUT1CHNG   0x20        /* button 1 changed? (1=yes) */
 #define MOVEMENT   0x40        /* mouse moved? (1=yes) */
```

FILES

/dev/mouse

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

null — the null file

DESCRIPTION

Data written on the null special file, */dev/null*, is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

prf — operating system profiler

DESCRIPTION

The special file */dev/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 */dev/prf* is a pseudo-device with no associated hardware.

FILES

/dev/prf

SEE ALSO

profiler(1M).

NAME

rtc — real time clock interface

DESCRIPTION

The rtc driver supports the real time clock chip, allowing it to be set with the correct local time, and allowing the time to be read from the chip.

Ioctl Calls

RTCRTIME

This call is used to read the local time from the real time clock chip. The argument to the *ioctl* is the address of a buffer of RTCNREG unsigned characters (RTCNREG is defined is **<sys/rtc.h>**). The ioctl will fill in the buffer with the contents of the chip registers. Currently, RTCNREG is 14, and the meanings of the byte registers are as follows:

Register	*Contents*
0	Seconds
1	Second alarm
2	Minutes
3	Minute alarm
4	Hours
5	Hour alarm
6	Day of week
7	Date of month
8	Month
9	Year
A	Status register A
B	Status register B
C	Status register C
D	Status register D

For further information on the functions of these registers, see the *IBM Personal Computer AT Technical Reference*.

RTCSTIME

This call is used to set the time into the real time clock chip. The argument to the *ioctl* is the address of a buffer of RTCNREGP unsigned characters (RTCNREGP is defined is **<sys/rtc.h>**.) These bytes should be the desired chip register contents. Currently, RTCNREGP is 10, representing registers 0-9 as shown above. Note that only user **root** (the superuser) may open the real-time clock device for writing, and that the RTCSTIME *ioctl* will fail for any user other than **root**.

FILES

/dev/rtc

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, contains enhancements to UNIX System V.

NAME

sasy – SunRiver asynchronous serial port

DESCRIPTION

The sasy driver supports both serial ports on the SunRiver Cygna 386 Station. If a station is not connected, an attempt to *open* its serial ports will fail. The ports can be programmed for speed (50-19200 baud), character length, and parity. Output speed is always the same as input speed. The ports behave as described in *termio*(7).

The asynchronous port is a character-at-a-time device for both input and output. This characteristic both limits the bandwidth which can be achieved over a line, and increases the interrupt loading on the central processor. In particular, file transfer programs such as *uucp*(1) may not function well at speeds over 9600 baud.

If the port was opened with the modem control bit present in the minor device number (see below), modem control will be enabled. If enabled, the driver will wait in *open* until Data Carrier Detect is present. Once opened, if Data Carrier Detect drops, the driver will return errors on any subsequent *read*s or *write*s of the asynchronous port by the user. If the port was opened as a controlling teletype, signal SIGHUP will be generated to the process which did the *open*.

The baud rates of the serial adapter programmable baud-rate generator do not correspond exactly with system baud rates. In particular, setting B0 will cause a disconnect, setting EXTA will set 19200 baud, and setting EXTB will set 38400 baud. It is not possible to directly set 2000, 3600, or 7200 baud.

Minor Device Numbers

The asynchronous port is a character device. The low-order 4 bits (bits 0-3) of the minor device number correspond to the station at which the port is located. Bit 4 enables modem control on the port. Thus, minor device 0 corresponds to the system board serial port with no modem control, while minor device 16 corresponds to the same port with modem control enabled.

FILES

/dev/ser*

SEE ALSO

asy(7), slp(7), termio(7).
signal(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

skd – SunRiver keyboard display driver

DESCRIPTION

The SunRiver console is composed of two separate pieces: the keyboard (see *keyboard*(7)) and the display. The keyboard interface is discussed separately in *keyboard*(7). The display is functionally equivalent to *display*(7) except for the differences described in this entry.

Ioctl Calls

The following *ioctl*s can be used with either the monochrome or color/graphics adapter:

KDDISPTYPE

This *ioctl* returns the following structure, which is pointed to with the argument that is passed:

```
struct kd_disparam {
 long type;                     /* display type */
 char *addr;                    /* display memory address */
 ushort ioaddr [ MKDIOADDR ];   /* valid I/O addresses */
};
```

The *type* member indicates the type of display and will be one of the following:

KD_MONO	1
KD_HERCULES	2
KD_CGA	3
KD_EGA	4
KD_VGA	5

The *addr* member indicates the physical address of the display. It will be either 0xF10000 in systems without cache memory and 0x8000F10000 in the case of systems with cache memory. The *ioaddr* member contains the global keyboard/display port list that is used for direct *in*s and *out*s to the SunRiver Station.

KDMAPDISP

This call allows one to have direct access to the display and I/O ports. It is especially useful in providing a way to do non-portable but fast graphics on the display. The following structure, defined in **<sys/kd.h>**, is pointed to by the argument to the *ioctl*:

```
struct kd_memloc {
   char     *vaddr;      /* virtual address to map to*/
   char     *physaddr;   /* physical address to map from */
   long     length;      /* size in bytes to map */
   long     ioflg;       /* enable I/O addresses if non-zero */
};
```

The *vaddr* argument is the linear address in the process where the display buffer will appear. This address must be on a page (4K byte) boundary. The *physaddr* argument is the physical address of the SunRiver Station. It must be between 0xF10000 and 0xF30000. It must also be on a page boundary. The *length* argument is the size of the display buffer that will be mapped in. It must be a multiple of 4K bytes. The *ioflg* argument tells whether (1) or not (0) to enable the global keyboard/display ports for direct access to the I/O ports similar to the KDENABIO and KDDISABIO *ioctl*s.

The memory that had existed at address *vaddr* for *length* bytes will be irretrievably deleted, and the current contents of the display buffer will be placed at those locations. It is necessary for the virtual terminal to be in process (**VT_PROCESS**) mode and for the display device to be in **KD_GRAPHICS** mode. One way of allocating the virtual memory in the user's address space that will be mapped to the screen is to call malloc requesting (*length* + 4096) bytes. Then using the address that malloc returns, round it up to the next page (4K byte) boundary and use the result as *vaddr*.

The *ioctl* will fail if the virtual terminal is not in process mode or if the display is not in **KD_GRAPHICS** mode [EACCES]. It will fail if any of the arguments are out of range or not properly aligned [EFAULT]. It will fail if the display is already mapped to [EIO].

KDMAPBIOS
: This call takes no arguments. It maps in the system BIOS to virtual address 0xC0000 in the user's address space.

KDMAPSUNRIV
: This *ioctl* is not implemented. It returns **−1**.

KDMACHENV
: This call returns information about the machine environment. The following structure, defined in **<sys/machenv.h>**, is returned by the *ioctl*:

```
struct machenv {
   unsigned    char    machine;
   unsigned    char    adapter;
   unsigned    char    drive;
   unsigned    char    special;
};
```

The only field currently being used is the **machine** field. This field can contain a **0** or the value **1**, which indicates that the computer in use is a COMPAQ.

FILES

/dev/st*

SEE ALSO

stty(1), console(7), display(7), keyboard(7), sasy(7), slp(7), termio(7), vt(7).
ioctl(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

slp – SunRiver parallel printer interface

DESCRIPTION

The slp driver is part of the INTERACTIVE UNIX System SunRiver Fiber Optic Station support facility. It provides the support for connecting a line printer to the back of the SunRiver Cygna 386 Station. The *close* waits until all output is competed before returning to the user. The lp driver allows only one process at a time to write to the printer. If it is already busy, an *open* for writing will return an error. However, the driver allows multiple *open*s to occur if they are *read-only*.

FILES

/dev/lp*
/dev/slp*

SEE ALSO

lp(7), sasy(7), skd(7).

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

streamio – STREAMS ioctl commands

SYNOPSIS

#include <stropts.h>
int ioctl fildes, command, arg)
int fildes, command;

DESCRIPTION

STREAMS [see *intro*(2)] ioctl commands are a subset of *ioctl*(2) system calls which perform a variety of control functions on *streams*. The arguments *command* and *arg* are passed to the file designated by *fildes* and are interpreted by the *stream head*. Certain combinations of these arguments may be passed to a module or driver in the *stream*.

fildes is an open file descriptor that refers to a *stream*. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl*, they are subject to the errors described there. In addition to those errors, the call will fail with *errno* set to EINVAL, without processing a control function, if the *stream* referenced by *fildes* is linked below a multiplexer, or if *command* is not a valid value for a *stream*.

Also, as described in *ioctl*, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the *stream head* containing an error value. This causes subsequent system calls to fail with *errno* set to this value.

COMMAND FUNCTIONS

The following *ioctl* commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH
: Pushes the module whose name is pointed to by *arg* onto the top of the current *stream*, just below the *stream head*. It then calls the open routine of the newly-pushed module. On failure, *errno* is set to one of the following values:

  [EINVAL] Invalid module name.

  [EFAULT] *arg* points outside the allocated address space.

  [ENXIO] Open routine of new module failed.

  [ENXIO] Hangup received on *fildes*.

I_POP
: Removes the module just below the *stream head* of the *stream* pointed to by *fildes*. *arg* should be 0 in an I_POP request. On failure, *errno* is set to one of the following values:

  [EINVAL] No module present in the *stream*.

  [ENXIO] Hangup received on *fildes*.

I_LOOK
: Retrieves the name of the module just below the *stream head* of the *stream* pointed to by *fildes*, and places it in

a null terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long. An [#include <sys/conf.h>] declaration is required. On failure, *errno* is set to one of the following values:

[EFAULT]	*arg* points outside the allocated address space.
[EINVAL]	No module present in *stream*.

I_FLUSH This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

FLUSHR	Flush read queues.
FLUSHW	Flush write queues.
FLUSHRW	Flush read and write queues.

On failure, *errno* is set to one of the following values:

[ENOSR]	Unable to allocate buffers for flush message due to insufficient STREAMS memory resources.
[EINVAL]	Invalid *arg* value.
[ENXIO]	Hangup received on *fildes*.

I_SETSIG Informs the *stream head* that the user wishes the kernel to issue the SIGPOLL signal [see *signal*(2) and *sigset*(2)] when a particular event has occurred on the *stream* associated with *fildes*. I_SETSIG supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

S_INPUT	A non-priority message has arrived on a *stream head* read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length.
S_HIPRI	A priority message is present on the *stream head* read queue. This is set even if the message is of zero length.
S_OUTPUT	The write queue just below the *stream head* is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.
S_MSG	A STREAMS signal message that contains the SIGPOLL signal has reached the front of the *stream head* read queue.

A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value S_HIPRI.

Processes that wish to receive SIGPOLL signals must explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same Stream, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further SIGPOLL signals. On failure, *errno* is set to one of the following values:

[EINVAL] *arg* value is invalid or *arg* is zero and process is not registered to receive the SIGPOLL signal.

[EAGAIN] Allocation of a data structure to store the signal request failed.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of I_SETSIG above. On failure, *errno* is set to one of the following values:

[EINVAL] Process not registered to receive the SIGPOLL signal.

[EFAULT] *arg* points outside the allocated address space.

I_FIND Compares the names of all modules currently present in the *stream* to the name pointed to by *arg*, and returns 1 if the named module is present in the *stream*. It returns 0 if the named module is not present. On failure, *errno* is set to one of the following values:

[EFAULT] *arg* points outside the allocated address space.

[EINVAL] *arg* does not contain a valid module name.

I_PEEK Allows a user to retrieve the information in the first message on the *stream head* read queue without taking the message off the queue. *arg* points to a *strpeek* structure which contains the following members:

```
struct strbuf      ctlbuf;
struct strbuf      databuf;
long               flags;
```

The *maxlen* field in the *ctlbuf* and *databuf strbuf* structures [see *getmsg*(2)] must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets *flags* to RS_HIPRI, I_PEEK will only look for a priority message on the *stream head* read queue.

L_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the *stream head* read queue, or if the RS_HIPRI flag was set in *flags* and a priority message was not present on the *stream head* read queue. It does not wait for a message to arrive. On return, *ctlbuf* specifies information in the control buffer, *databuf* specifies information in the data buffer, and *flags* contains the value 0 or RS_HIPRI. On failure, *errno* is set to one of the following values:

[EFAULT] *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.

[EBADMSG] Queued message to be read is not valid for L_PEEK

L_SRDOPT Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

RNORM Byte-stream mode, the default.

RMSGD Message-discard mode.

RMSGN Message-nondiscard mode.

Read modes are described in *read*(2). On failure, *errno* is set to the following value:

[EINVAL] *arg* is not one of the above legal values.

L_GRDOPT Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in *read*(2). On failure, *errno* is set to the following value:

[EFAULT] *arg* points outside the allocated address space.

L_NREAD Counts the number of data bytes in data blocks in the first message on the *stream head* read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the *stream head* read queue. For example, if zero is returned in *arg*, but the *ioctl* return value is greater than zero, this indicates that a zero-length message is next on the queue. On failure, *errno* is set to the following value:

[EFAULT] *arg* points outside the allocated address space.

L_FDINSERT Creates a message from user specified buffer(s), adds information about another *stream* and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

arg points to a *strfdinsert* structure which contains the following members:

```
struct strbuf          ctlbuf;
```

```
struct strbuf      databuf;
long               flags;
int                fildes;
int                offset;
```

The *len* field in the *ctlbuf strbuf* structure [see *putmsg*(2)] must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fildes* in the *strfdinsert* structure specifies the file descriptor of the other *stream*. *offset*, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where I_FDINSERT will store a pointer. This pointer will be the address of the read queue structure of the driver for the *stream* corresponding to *fildes* in the *strfdinsert* structure. The *len* field in the *databuf strbuf* structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

flags specifies the type of message to be created. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to RS_HIPRI. For non-priority messages, I_FDINSERT will block if the *stream* write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NDELAY is set. Instead, it fails and sets *errno* to EAGAIN.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether O_NDELAY has been specified. No partial message is sent. On failure, *errno* is set to one of the following values:

[EAGAIN] A non-priority message was specified, the O_NDELAY flag is set, and the *stream* write queue is full due to internal flow control conditions.

[ENOSR] Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.

[EFAULT] *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.

[EINVAL] One of the following: *fildes* in the *strfdinsert* structure is not a valid, open *stream* file descriptor; the size of a pointer plus *offset* is greater than the *len* field for the buffer specified through *ctlptr*; *offset* does not specify a properly aligned location in the data buffer; an undefined value is stored in *flags*.

[ENXIO] Hangup received on *fildes* of the *ioctl* call or *fildes* in the *strfdinsert* structure.

[ERANGE] The *len* field for the buffer specified through *databuf* does not fall within the range specified by the maximum and minimum packet sizes of the topmost *stream* module, or the *len* field for the buffer specified through *databuf* is larger than the maximum configured size of the data part of a message, or the *len* field for the buffer specified through *ctlbuf* is larger than the maximum configured size of the control part of a message.

I_FDINSERT can also fail if an error message was received by the *stream head* of the *stream* corresponding to *fildes* in the *strfdinsert* structure. In this case, *errno* will be set to the value in the message.

I_STR Constructs an internal STREAMS ioctl message from the data pointed to by *arg* and sends that message downstream.

This mechanism is provided to send user *ioctl* requests to downstream modules and drivers. It allows information to be sent with the *ioctl* and will return to the user any information sent upstream by the downstream recipient. I_STR blocks until the system responds with either a positive or negative acknowledgment message or until the request "times out" after some period of time. If the request times out, it fails with *errno* set to ETIME.

At most, one I_STR can be active on a *stream*. Further I_STR calls will block until the active I_STR completes at the *stream head*. The default timeout interval for these requests is 15 seconds. The O_NDELAY [see *open*(2)] flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioctl* structure which contains the following members:

```
int     ic_cmd;      /* downstream command */
int     ic_timout;   /* ACK/NAK timeout */
int     ic_len;      /* length of data arg */
char    *ic_dp;      /* ptr to data arg */
```

ic_cmd is the internal ioctl command intended for a downstream module or driver; and *ic_timout* is the number of seconds (-1 = infinite, 0 = use default, >0 = as specified) an I_STR request will wait for acknowledgment before timing out. *ic_len* is the number of bytes in the data argument and *ic_dp* is a pointer to the data argument. The *ic_len* field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the *stream* can return).

The *stream head* will convert the information pointed to by the *strioctl* structure to an internal ioctl command message and send it downstream. On failure, *errno* is set to one of the following values:

[ENOSR] Unable to allocate buffers for the *ioctl* message due to insufficient STREAMS memory resources.

[EFAULT] *arg* points, or the buffer area specified by *ic_dp* and *ic_len* (separately for data sent and data returned) is, outside the allocated address space.

[EINVAL] *ic_len* is less than 0 or *ic_len* is larger than the maximum configured size of the data part of a message or *ic_timout* is less than -1.

[ENXIO] Hangup received on *fildes*.

[ETIME] A downstream *ioctl* timed out before acknowledgment was received.

An I_STR can also fail while waiting for an acknowledgment if a message indicating an error or a hangup is received at the *stream head*. In addition, an error code can be returned in the positive or negative acknowledgment message, in the event the ioctl command sent downstream fails. For these cases, I_STR will fail with *errno* set to the value in the message.

I_SENDFD Requests the *stream* associated with *fildes* to send a message, containing a file pointer, to the *stream head* at the other end of a *stream* pipe. The file pointer corresponds to *arg*, which must be an integer file descriptor.

I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue [see *intro*(2)] of the *stream head* at the other end of the *stream* pipe to which it is connected. On failure, *errno* is set to one of the following values:

[EAGAIN] The sending *stream* is unable to allocate a message block to contain the file pointer.

[EAGAIN] The read queue of the receiving *stream head* is full and cannot accept the message sent by I_SENDFD.

[EBADF] *arg* is not a valid, open file descriptor.

[EINVAL] *fildes* is not connected to a *stream* pipe.

[ENXIO] Hangup received on *fildes*.

I_RECVFD Retrieves the file descriptor associated with the message sent by an I_SENDFD *ioctl* over a *stream* pipe. *arg* is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

fd is an integer file descriptor. *uid* and *gid* are the user id and group id, respectively, of the sending *stream*.

If O_NDELAY is not set [see *open*(2)], I_RECVFD will block until a message is present at the *stream head*. If O_NDELAY is set, I_RECVFD will fail with *errno* set to EAGAIN if no message is present at the *stream head*.

If the message at the *stream head* is a message sent by an I_SENDFD, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the *strrecvfd* structure. The structure is copied into the user data buffer pointed to by *arg*. On failure, *errno* is set to one of the following values:

[EAGAIN] A message was not present at the *stream head* read queue, and the O_NDELAY flag is set.

[EBADMSG] The message at the *stream head* read queue was not a message containing a passed file descriptor.

[EFAULT] *arg* points outside the allocated address space.

[EMFILE] NOFILES file descriptors are currently open.

[ENXIO] Hangup received on *fildes*.

The following two commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK Connects two *streams*, where *fildes* is the file descriptor of the *stream* connected to the multiplexing driver, and *arg* is the file descriptor of the *stream* connected to another driver. The *stream* designated by *arg* gets connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgment message to the *stream head* regarding the linking operation. This call returns a multiplexer ID number (an identifier used to disconnect the multiplexer, see I_UNLINK) on success, and a -1 on failure. On failure, *errno* is set to one of the following values:

[ENXIO] Hangup received on *fildes*.

[ETIME] Time out before acknowledgment message was received at *stream head*.

[EAGAIN] Temporarily unable to allocate storage to perform the I_LINK.

[ENOSR] Unable to allocate storage to perform the I_LINK due to insufficient STREAMS memory resources.

[EBADF] *arg* is not a valid, open file descriptor.

[EINVAL] *fildes stream* does not support multiplexing.

[EINVAL] *arg* is not a *stream*, or is already linked under a multiplexer.

[EINVAL] The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given *stream head* is linked into a multiplexing configuration in more than one place.

An I_LINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgment message. For these cases, I_LINK will fail with *errno* set to the value in the message.

I_UNLINK Disconnects the two *streams* specified by *fildes* and *arg*. *fildes* is the file descriptor of the *stream* connected to the multiplexing driver. *fildes* must correspond to the *stream* on which the *ioctl* I_LINK command was issued to link the *stream* below the multiplexing driver. *arg* is the multiplexer ID number that was returned by the I_LINK. If *arg* is -1, then all Streams which were linked to *fildes* are disconnected. As in I_LINK, this command requires the multiplexing driver to acknowledge the unlink. On failure, *errno* is set to one of the following values:

[ENXIO] Hangup received on *fildes*.

[ETIME] Time out before acknowledgment message was received at *stream head*.

[ENOSR] Unable to allocate storage to perform the I_UNLINK due to insufficient STREAMS memory resources.

[EINVAL] *arg* is an invalid multiplexer ID number or *fildes* is not the *stream* on which the I_LINK that returned *arg* was performed.

An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgment message. For these cases, I_UNLINK will fail with *errno* set to the value in the message.

SEE ALSO

close(2), fcntl(2), intro(2), ioctl(2), open(2), read(2), getmsg(2), poll(2), putmsg(2), signal(2), sigset(2), write(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.
STREAMS Programmer's Guide.
STREAMS Primer.

DIAGNOSTICS

Unless specified otherwise above, the return value from *ioctl* is 0 upon success and -1 upon failure with *errno* set as indicated.

NAME

sxt — pseudo-device driver

DESCRIPTION

The special file */dev/sxt* is a pseudo-device driver that interposes a discipline between the standard *tty* line disciplines and a real device driver. The standard disciplines manipulate *virtual tty* structures (channels) declared by the */dev/sxt* driver. */dev/sxt* acts as a discipline manipulating a *real tty* structure declared by a real device driver. The */dev/sxt* driver is currently only used by the *shl*(1) command.

Virtual ttys are named by inodes in the subdirectory **/dev/sxt** and are allocated in groups of up to eight. To allocate a group, a program should exclusively open a file with a name of the form **/dev/sxt/??0** (channel 0) and then execute a SXTIOCLINK *ioctl* call to initiate the multiplexing.

Only one channel, the *controlling* channel, can receive input from the keyboard at a time; others attempting to read will be blocked.

There are two groups of *ioctl*(2) commands supported by *sxt*. The first group contains the standard *ioctl* commands described in *termio*(7), with the addition of the following:

TIOCEXCL — Set *exclusive use* mode: no further opens are permitted until the file has been closed.

TIOCNXCL — Reset *exclusive use* mode: further opens are once again permitted.

The second group are commands to *sxt* itself. Some of these may only be executed on channel 0.

SXTIOCLINK — Allocate a channel group and multiplex the virtual ttys onto the real tty. The argument is the number of channels to allocate. This command may only be executed on channel 0. Possible errors include:

- EINVAL — The argument is out of range.
- ENOTTY — The command was not issued from a real tty.
- ENXIO — *linesw* is not configured with *sxt*.
- EBUSY — An SXTIOCLINK command has already been issued for this real *tty*.
- ENOMEM — There is no system memory available for allocating the virtual tty structures.
- EBADF — Channel 0 was not opened before this call.

SXTIOCSWTCH — Set the controlling channel. Possible errors include:

- EINVAL — An invalid channel number was given.

	EPERM	The command was not executed from channel 0.

SXTIOCWF
: Cause a channel to wait until it is the controlling channel. This command will return the error *EINVAL* if an invalid channel number is given.

SXTIOCUBLK
: Turn off the **loblk** control flag in the virtual tty of the indicated channel. The error *EINVAL* will be returned if an invalid number or channel 0 is given.

SXTIOCSTAT
: Get the status (blocked on input or output) of each channel and store in the *sxtblock* structure referenced by the argument. The error *EFAULT* will be returned if the structure cannot be written.

SXTIOCTRACE
: Enable tracing. Tracing information is written to the console. This command has no effect if tracing is not configured.

SXTIOCNOTRACE
: Disable tracing. This command has no effect if tracing is not configured.

FILES

/dev/sxt/??[0-7] Virtual tty devices

SEE ALSO

shl(1), stty(1), termio(7).
ioctl(2), open(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

tape – cartridge tape storage

DESCRIPTION

Cartridge tape devices and disks are supported by the High Performance Disk Driver (see *hpdd*(7)). Magnetic tape cartridges provide for backup and exchange of data normally stored on fixed disks. The tape driver supports the Archive Viper series of Small Computer System Interface (SCSI) tape drives, models 2125S, and 2150S. These drives must be equipped with firmware EPROMs of at least revision 6 level (for the 2125S) or revision 4 level (for the 2150S). The firmware revision number is located on a sticker on the EPROM, on the component side of the tape drive main printed circuit board. The drive should be configured for an otherwise unused SCSI ID number and should be jumpered for a "Disconnect Size" of no more than 16 KB. See the *Archive Viper Product Manual* for more information.

The tape driver can *read* 60 MB QIC-24 tapes, and *read* or *write* 125 MB QIC-120 or 150 MB QIC-150 (2150S drive only) tapes. Note that *only* DC600XTD tapes are certified for 150 MB recording, and that DC600A tapes are certified for 125 MB recording.

Cartridge tapes are treated as character devices. However, they actually record fixed-size (512-byte) data blocks. If a *write* call is made to the driver with a length that is not a multiple of 512 bytes, the last block is automatically padded with zeroes. A *read* call with a length that is not a multiple of 512 bytes will result in data from the last block being discarded by the driver. No attempt is made by the driver to allocate large blocks of memory to insure efficient streaming operation of the drive, although the drives themselves contain 64KB buffers. The *dd*(1) command can be used as a filter to provide buffering between a command and the driver, or the –C option of the *cpio*(1) command can be used to allocate large tape buffers.

Device Names and Numbers

All devices handled by the *tape* driver have the same major number (currently 41). Each tape drive has a span of 16 minor numbers associated with it. The interpretation of the minor numbers for tape devices is described in the file **/usr/include/sys/gentape.h**. Bits 4-7 (the high-order nibble of the minor number) indicate the drive number. Thus, minor devices 0-15 refer to the first drive, devices 16-31 refer to the second drive, etc. For each drive, bit 3 should be set if the tape should be rewound when it is closed. This is set for the **/dev/tape** device and is the normal usage. This type of device allows a single file to be written on a tape. If bit 3 is *not* set, the tape will *not* be rewound when closed. (See below for a complete description of tape behavior on closing the device.) This is reset for the **/dev/ntape** device and allows multiple files, separated by *file marks* (described below) to exist on a single tape. Bit 2 of the minor number should be set if positioning operations (rewinding, seeking a file mark) should return immediately after the requests initiating them have been sent to the drive. If this bit is not set, the driver will wait for such operations to be completed before returning control to user programs. Note that the *open* system call always waits for any long-term operation that may be in progress to complete before it attempts to open the drive.

Bits 0 and 1 can indicate a tape density code, although this value is ignored. The values are as follows:

0	<Default (see below)>
1	60 MB tape
2	120 MB tape
3	150 MB tape

The *default* value indicates that the drive should determine the density based on the type of cartridge used.

File Marks and End-of-Medium

A special marker, known as a file mark or End of File (EOF) is normally written at the end of each data file written to the tape. If a file mark is detected in the middle of a multi-block *read* operation, a short block count will be returned. Subsequent *read* operations (or any *read* that encounters a file mark prior to any data blocks) will return EOF. If an attempt is made to perform a *write* operation that gets closer than about 1 MB to the physical end of the tape, all data for that command will be written to the tape. Subsequent *write* operations will return End of Medium.

Tape Position on Close

When the tape device is closed, its position depends on several factors: whether it is to be rewound (bit 3 in minor device number), whether it was opened for a *read* or a *write*, whether it is currently positioned at a file mark or the End of Medium, and whether the last operation performed was an *ioctl* (described below).

If the last operation performed on the drive was a standard *read* or *write* (or simply open), the type of open done to the drive is checked. If it was opened for *write*, a file mark is written. If it was opened for *read* and a file mark was not the last thing read, and the tape is not to be rewound on closing, a *Seek to End-of-File* is initiated. If the last operation performed on the drive was an *ioctl* call, *no* further positioning or file mark writing is performed. Finally, the tape is rewound if bit 3 of the minor device number is set.

Ioctl Calls Supported

The following *ioctl* calls are defined in the file **/usr/include/sys/gentape.h** . Unless stated otherwise, the *arg* value of the *ioctl* call is ignored.

TC_SEOF

This call takes a count as an argument and seeks to the *arg*th file mark on the tape.

TC_REWIND

This call causes the tape to be rewound to the beginning of tape point.

TC_WFM

This call takes a count as an argument and writes *count* file marks onto the tape. This command also causes any unwritten data in the drive's cache to be written to the tape. An arg of 0 is valid and simply causes the drive to flush its cache to tape.

TC_ERASE

This call causes the entire tape to be erased. This command is only valid when the tape is at the "beginning of tape" point.

TC_RETENSION

This call causes the tape to be shuttled from beginning to end and back again, causing clean packing of the tape against the hubs. New tapes should be retensioned prior to use, as should tapes that have small portions of their length used extensively.

TC_WAIT

This call will not return until a previously issued long-term command (TC_SEOF, TC_REWIND, TC_ERASE, or TC_RETENSION) has completed. This command only makes sense when used on a tape device with bit 2 of the minor device number (for immediate return on long-term operations) set.

TC_GETPARMS

This call requires a pointer at **struct tc_parms** (see the file **/usr/include/sys/gentape.h**) as an argument. It will return with information about the drive and the tape currently loaded in it. Note that this information may not be meaningful until the drive has actually attempted *read* or *write* operations to the tape.

Configuration

In order to use the tape device, it must be configured into the kernel. To do this, the High Performance Disk Driver configuration file (**/etc/conf/pack.d/gendev/space.c**) must contain an external definition such as the following:

extern int tape_init();

and the line:

tape_init,

must appear in the initialization of the **gdev_init_routines** table defined near the end of that file. The entry for **tape_init** must appear prior to the NULL line in the initialization. This instructs the High Performance Disk Driver to invoke the initialization routine for tapes when the system boots.

In addition, the file **/etc/conf/pack.d/tape/space.c** contains a configuration definition for each tape-supporting controller board permitted.

Note also that a driver module for a tape-supporting controller board must also be enabled when the kernel is built, so that this low-level module will be included.

FILES

/usr/include/sys/gentape.h
/etc/conf/pack.d/gendev/space.c
/etc/conf/pack.d/tape/space.c
/etc/conf/cf.d/mdevice
/dev/tape
/dev/ntape

SEE ALSO

cpio(1), dd(1), aha1540(7), athd(7), hpdd(7), tmc8x0(7).

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

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 terminal 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, the buffer is flushed and 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 (CTRL- 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 (CTRL-Z or ASCII SUB) is used by the job control facility, *shl,* to change the current layer to the control layer.

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 (CTRL-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 cannot be changed or escaped.

EOL (ASCII NUL) is an additional line delimiter, like NL. It is not normally used.

EOL2 is another additional line delimiter.

STOP (CTRL-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 (CTRL-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 XENIX extension ioctl calls use the following structures that are defined in **<ttold.h>**:

```
struct   sgttyb {
         char             sg_ispeed;
         char             sg_ospeed;
         char             sg_erase;
         char             sg_kill;
         short            sg_flags;
};

struct   tc {
         char             t_intrc;
         char             t_quitc;
         char             t_startc;
         char             t_stopc;
         char             t_eofc;
         char             t_brkc;
};
```

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	

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.

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 lower case 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_cflag* 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 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
B19200	0000016	19200 baud
EXTA	0000016	External A
B38400	0000017	38400 baud
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.
RCV1EN	0010000	
XMT1EN	0020000	
LOBLK	0040000	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.

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.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The *c_lflag* 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 following form:

```
ioctl(fildes, command, arg);
```

where *fildes* is the file descriptor of the opened tty device, *command* is the ioctl type, and *arg* is a buffer or structure depending on the ioctl call used.

For the following calls using this form, the *arg* is of type:

```
struc termio *arg;
```

TCGETA
: Gets the parameters associated with the terminal and store them in the *termio* structure referenced by *arg*.

TCSETA
: Sets the parameters associated with the terminal from the structure referenced by *arg*. The change is immediate.

TCSETAW
: Waits for the output to drain before setting the new

parameters. This form should be used when changing parameters that will affect output.

TCSETAF
: Waits for the output to drain, then flush the input queue and set the new parameters.

For the following calls using this form, the *arg* parameter must be a pointer to a buffer of size 1024:

LDSMAP
: Sets the input/output mapping on a channel.

LDGMAP
: Gets the input/output mapping on a channel.

LDNMAP
: Turns off the input/output mapping on a channel. The *arg* for this call must be a NUL pointer.

For the following ioctl calls, the *arg* parameter is of type:
struc tc *arg;

TIOCGETC
: Gets the current settings of special characters.

TIOCSETC
: Changes the settings for special characters.

The TIOCSETN ioctl call requires a different *arg* type. The *arg* for this call must be of type:
struc sgttyb *arg;

TIOCSETN
: Sets the parameters associated with the terminal, but does not delay or flush the input.

For the following ioctl calls, the *arg* is of type NUL:

TIOCEXCL
: Sets exclusive use mode: no futher opens are permitted until the file has been closed.

TIONEXCL
: Resets exclusive use mode: further opens are once again permitted.

TIOCHPCL
: When the file is closed for the last time, hang up the terminal.

TIOCFLUSH
: All characters waiting in the input or output queues are flushed.

FIORDCHK
: Return a non-zero value if there are characters on the input.

Additional *ioctl*(2) calls have the form:

ioctl (fildes, command, arg)

where *arg* is of type int or a pointer to an unsigned short, depending on the ioctl call used.

For the following ioctl calls using this form, the *arg* is of type int:

TCSBRK	Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).
TCXONC	Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.
TCFLSH	If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

For the following ioctl calls using this form, the *arg* is of type:
unsigned short *arg;

TIOCGETD
Gets the line discipline associated with the terminal.

TIOCSETD
Sets the line discipline associated with the terminal.

The DIOCSETP and DIOCGETP ioctl calls are no-op calls which have been retained for backwards compatibility.

FILES

/dev/tty*

SEE ALSO

stty(1).
fork(2), ioctl(2), setpgrp(2), signal(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

timod – Transport Interface cooperating STREAMS module

DESCRIPTION

timod is a STREAMS module for use with the Transport Interface (TI) functions of the Network Services library. The *timod* module converts a set of *ioctl*(2) calls into STREAMS messages that may be consumed by a transport protocol provider which supports the Transport Interface. This allows a user to initiate certain TI functions as atomic operations.

The *timod* module must only be pushed (see *Streams Primer*) onto a *stream* terminated by a transport protocol provider which supports the TI.

All STREAMS messages, with the exception of the message types generated from the *ioctl* commands described below, will be transparently passed to the neighboring STREAMS module or driver. The messages generated from the following *ioctl* commands are recognized and processed by the *timod* module. The format of the *ioctl* call is:

```
#include <sys/stropts.h>
                 -
                 -
struct strioctl strioctl;
                 -
                 -
strioctl.ic_cmd = cmd;
strioctl.ic_timeout = INFTIM;
strioctl.ic_len = size;
strioctl.ic_dp = (char *)buf

ioctl(fildes, I_STR, &strioctl);
```

where, on issuance, *size* is the size of the appropriate TI message to be sent to the transport provider and on return, *size* is the size of the appropriate TI message from the transport provider in response to the issued TI message. *buf* is a pointer to a buffer large enough to hold the contents of the appropriate TI messages. The TI message types are defined in **<sys/tihdr.h>**. The possible values for the *cmd* field are:

TI_BIND — Bind an address to the underlying transport protocol provider. The message issued to the TI_BIND *ioctl* is equivalent to the TI message type T_BIND_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_BIND_ACK.

TI_UNBIND — Unbind an address from the underlying transport protocol provider. The message issued to the TI_UNBIND *ioctl* is equivalent to the TI message type T_UNBIND_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_OK_ACK.

TI_GETINFO — Get the TI protocol specific information from the transport protocol provider. The message issued to the TI_GETINFO *ioctl* is equivalent to the TI message type T_INFO_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_INFO_ACK.

TI_OPTMGMT Get, set, or negotiate protocol specific options with the transport protocol provider. The message issued to the TI_OPTMGMT *ioctl* is equivalent to the TI message type T_OPTMGMT_REQ, and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_OPTMGMT_ACK.

FILES

<sys/timod.h>
<sys/tiuser.h>
<sys/tihdr.h>
<sys/errno.h>

SEE ALSO

tirdwr(7).
STREAMS Primer.
STREAMS Programmer's Guide.
Network Programmer's Guide.

DIAGNOSTICS

If the *ioctl* system call returns with a value greater than 0, the lower 8 bits of the return value will be one of the TI error codes as defined in **<sys/tiuser.h>**. If the TI error is of type TSYSERR, then the next 8 bits of the return value will contain an error as defined in **<sys/errno.h>** [see *intro*(2)].

NAME

tirdwr – Transport Interface read/write interface STREAMS module

DESCRIPTION

tirdwr is a STREAMS module that provides an alternate interface to a transport provider which supports the Transport Interface (TI) functions of the Network Services library (see Section 3N). This alternate interface allows a user to communicate with the transport protocol provider using the *read*(2) and *write*(2) system calls. The *putmsg*(2) and *getmsg*(2) system calls may also be used. However, *putmsg* and *getmsg* can only transfer data messages between user and *stream*.

The *tirdwr* module must only be pushed [see I_PUSH in *streamio*(7)] onto a *stream* terminated by a transport protocol provider which supports the TI. After the *tirdwr* module has been pushed onto a *stream*, none of the Transport Interface functions can be used. Subsequent calls to TI functions will cause an error on the *stream*. Once the error is detected, subsequent system calls on the *stream* will return an error with *errno* set to EPROTO.

The following are the actions taken by the *tirdwr* module when pushed on the *stream*, popped [see I_POP in *streamio*(7)] off the *stream*, or when data passes through it.

push – When the module is pushed onto a *stream*, it will check any existing data destined for the user to ensure that only regular data messages are present. It will ignore any messages on the *stream* that relate to process management, such as messages that generate signals to the user processes associated with the *stream*. If any other messages are present, the I_PUSH will return an error with *errno* set to EPROTO.

write – The module will take the following actions on data that originated from a *write* system call:

- All messages with the exception of messages that contain control portions (see the *putmsg* and *getmsg* system calls) will be transparently passed onto the module's downstream neighbor.
- Any zero length data messages will be freed by the module and they will not be passed onto the module's downstream neighbor.
- Any messages with control portions will generate an error, and any further system calls associated with the *stream* will fail with *errno* set to EPROTO.

read – The module will take the following actions on data that originated from the transport protocol provider:

- All messages with the exception of those that contain control portions (see the *putmsg* and *getmsg* system calls) will be transparently passed onto the module's upstream neighbor.
- The action taken on messages with control portions will be as follows:

+ Messages that represent expedited data will generate an error. All further system calls associated with the *stream* will fail with *errno* set to EPROTO.

+ Any data messages with control portions will have the control portions removed from the message prior to passing the message to the upstream neighbor.

+ Messages that represent an orderly release indication from the transport provider will generate a zero length data message, indicating the end of file, which will be sent to the reader of the *stream*. The orderly release message itself will be freed by the module.

+ Messages that represent an abortive disconnect indication from the transport provider will cause all further *write* and *putmsg* system calls to fail with *errno* set to ENXIO. All further *read* and *getmsg* system calls will return zero length data (indicating end of file) once all previous data has been read.

+ With the exception of the above rules, all other messages with control portions will generate an error and all further system calls associated with the *stream* will fail with *errno* set to EPROTO.

– Any zero length data messages will be freed by the module and they will not be passed onto the module's upstream neighbor.

pop — When the module is popped off the *stream* or the *stream* is closed, the module will take the following action:

– If an orderly release indication has been previously received, then an orderly release request will be sent to the remote side of the transport connection.

SEE ALSO

streamio(7), timod(7).
intro(2), getmsg(2), putmsg(2), read(2), write(2), intro(3) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*
STREAMS Primer.
STREAMS Programmer's Guide.
Network Programmer's Guide.

NAME

tmc8x0 — low-level controller module

DESCRIPTION

The *tmc8x0* module provides low-level interface routines between the High Performance Disk Driver (see *hpdd*(7)) and the Future Domain TMC-830 and TMC-841 Intelligent Small Computer System Interface (SCSI) Host Adapters. It can be configured for disk or streaming tape support for one or more host adapter boards, each of which must be the sole initiator on a SCSI bus. The low-level code determines if the adapter is, in fact, present at the configured address (see *disk*(7) and *tape*(7)) and what types of devices are attached to it. It handles simultaneous requests for all devices, overlapping operations where the devices will permit.

For information on configuring a kernel to include the *tmc8x0* module, refer to the "INTERACTIVE UNIX Operating System Maintenance Procedures."

Board Configuration

The default configuration described in your Future Domain TMC-830 and TMC-841 user's manual should be used for standard configurations of the system. If more than one board is to be used in a single system, each must at least occupy a different set of I/O address ranges and use a different DMA channel. Use of a different interrupt level for each board is encouraged, although not required.

SEE ALSO

aha1540(7), athd(7), disk(7), hpdd(7), tape(7).
"INTERACTIVE UNIX Operating System Maintenance Procedures" in the *INTERACTIVE UNIX Operating System Guide*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

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*

SEE ALSO

console(7).

NAME

vna — UnTerminal video network adapter

DESCRIPTION

The UnTerminal video network adapter (VNA) is composed of two separate pieces: the keyboard (see *keyboard*(7)) and the display. The keyboard interface is discussed separately in *keyboard*(7). The display is functionally equivalent to *display*(7) except for the differences described in this entry.

Ioctl Calls

The following *ioctl*s can be used with the VNA adapter:

KDDISPTYPE

This *ioctl* returns the following structure, which is pointed to with the argument that is passed:

```
struct kd_disparam {
 long type;                        /* display type */
 char *addr;                       /* display memory address */
 ushort ioaddr [ MKDIOADDR ];      /* valid I/O addresses */
};
```

The *type* member indicates the type of display and will be one of the following:

KD_HERCULES	2
KD_VGA	5

The *addr* member indicates the physical address of the display. This address corresponds to switch settings on the board and typically ranges from 0xF00,000 to 0xFF0,000. The *ioaddr* member contains the global keyboard/display port list that is used for direct *in*s and *out*s to the UnTerminal remote station.

KDMAPDISP

This call allows one to have direct access to the display and I/O ports. It is especially useful in providing a way to do non-portable but fast graphics on the display. The following structure, defined in **<sys/kd.h>**, is pointed to by the argument to the *ioctl*:

```
struct kd_memloc {
   char    *vaddr;      /* virtual address to map to*/
   char    *physaddr;   /* physical address to map from */
   long    length;      /* size in bytes to map */
   long    ioflg;       /* enable I/O addresses if non-zero */
};
```

The *vaddr* argument is the linear address in the process where the display buffer will appear. This address must be on a page (4K byte) boundary. The *physaddr* argument is the physical address of the UnTerminal remote station. It should correspond to the *addr* value returned by the ioctl KDDISPTYPE. The *length* argument is the size of the display buffer that will be mapped in. It must be a multiple of 4K bytes.

The *ioflg* argument tells whether (1) or not (0) to enable the global keyboard/display ports for direct access to the I/O ports similar to the KDENABIO and KDDISABIO *ioctl*s.

The memory that had existed at address *vaddr* for *length* bytes will be irretrievably deleted, and the current contents of the display buffer will be placed at those locations. It is necessary for the virtual terminal to be in process (VT_PROCESS) mode and for the display device to be in KD_GRAPHICS mode. One way of allocating the virtual memory in the user's address space that will be mapped to the screen is to call malloc requesting (*length* + 4096) bytes. Then using the address that malloc returns, round it up to the next page (4K byte) boundary and use the result as *vaddr*.

The *ioctl* will fail if the virtual terminal is not in process mode or if the display is not in KD_GRAPHICS mode [EACCES]. It will fail if any of the arguments are out of range or not properly aligned [EFAULT]. It will fail if the display is already mapped to [EIO].

FILES

/dev/vna*

SEE ALSO

stty(1), console(7), display(7), iolp(7), iona(7), keyboard(7), termio(7), vt(7).
ioctl(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

vt – virtual terminal management

DESCRIPTION

The virtual terminal (VT) device driver is a layer of management functions that provides the facilities to support and switch between up to eight screen faces on each physical device. Terminal or display device drivers that have been written to take advantage of this facility can therefore present multiple VTs on a single physical device. The correspondence between physical and virtual terminals is determined using the minor device number of the physical device, with the bottom five bits selecting the physical device and the top three bits selecting the virtual terminal.

Virtual terminals are accessed in exactly the same way as any other device. The *open*(2) system call is used to open the virtual terminal, and *read*(2), *write*(2), and *ioctl*(2) are used in the normal way and support all the functionality of the underlying device. In addition, some VT-specific *ioctls* are provided as described below.

Virtual terminals provide the link between different screen faces and the device. The virtual terminal that corresponds to the currently visible screen face is said to be the active virtual terminal. The active VT is the one that input from the device will be directed to, and any device-specific modes that can change on a per VT basis will be set to the characteristics associated with the active VT.

Open virtual terminals on a device are placed on a "ring," with the active VT always being the VT on the top of the ring. The ring can be cycled through via a "hot key" that is specific to the underlying device driver. The first open of a VT causes it to be placed at the top of the ring and become the active VT. The last close on a VT causes it to be removed from the ring, and if this was the active VT, the previous VT on the ring becomes the active one.

Virtual terminal switching can be done in two different modes: automatically (VT_AUTO) on receipt of a "hot key," or under control of the process owning the VT (VT_PROCESS). In the first case, the process associated with the VT knows nothing about the switch, and saving and restoring the device are handled entirely by the underlying device driver and the virtual terminal manager. So that the ordinary process need not be concerned with it, this is the default VT mode; switching will be done automatically.

In process-controlled switch mode, when a switch "hot key" is sent, the process owning the VT is sent a signal (*relsig*) that it has specified to the VT manager (see *sigset*(2)). This signal requests the process to release the physical device. The VT manager then awaits the VT_RELDISP *ioctl* from the process. If the process refuses to release the device (in which case the switch does not occur), it does a VT_RELDISP *ioctl* with an argument of 0 (zero). If a predefined time limit expires before the VT_RELDISP *ioctl* is received from the process owning the VT, the VT manager behaves as if an *ioctl* indicating refusal was received. If the process desires to release the device, it should save the device state (keyboard, display, and I/O registers) and then do the VT_RELDISP *ioctl* with an argument of 1. At this time the switch will occur.

The ring of active VTs can contain intermixed auto mode and process control mode VTs. When an auto mode process becomes active, the underlying device driver and the virtual terminal manager take care of restoring the device. Process control mode processes will be sent a signal that they have specified (*acqsig*) when they become the active VT. The process should restore the device state (keyboard, display, and I/O registers) and then do the VT_RELDISP *ioctl* with an argument of VT_ACKACQ. This completes the VT switching protocol.

Some device drivers may support a forced switch mode, in which case an alternate hotkey sequence will cause the driver to force a switch to the next VT even if a normal switch is refused. The driver does the forced switch and the VT manager signals the VT that it has been forced out.

Ioctl Calls

The following *ioctl*s apply to any device that supports VTs.

VT_OPENQRY

This call is used to find an available VT. The argument to the *ioctl* is a pointer to a long. The long will be filled in with the number of the first available VT that no other process has open (and hence, is available to be opened). If there are no available VTs, then −1 will be filled in.

VT_GETMODE

This call is used to determine what mode the VT is currently in, either VT_AUTO or VT_PROCESS. The argument to the *ioctl* is the address of the following structure, as defined in **<sys/vt.h>**.

```
struct vt_mode {
     char    mode;      /* VT mode */
     char    waitv;     /* if non-zero, hang on writes when
                        not active */
     short   relsig;    /* signal to use for release request */
     short   acqsig;    /* signal to use for display acquired */
     short   frsig;     /* signal to use for forced release */
}

/* Virtual Terminal Modes */
#define     VT_AUTO        0/* automatic VT switching */
#define     VT_PROCESS     1/* process controls switching */
```

The structure will be filled in with the current value for each field.

VT_SETMODE

This call is used to set the VT mode. The argument to the *ioctl* is a pointer to a **vt_mode** structure, as defined above. The structure should be filled in with the desired VT mode and whether or not to block on writes when not active. If process-control mode is specified then the signals that should be used to communicate with the process should be specified. If any of the signals are not specified (value is zero), then the default for that signal will be used (SIGUSR1 for *relsig* and *acqsig* and SIGUSR2 for *frsig*).

VT_RELDISP

This call is used to tell the VT manager if the display has been released or if the process has refused to release the display. An argument of 0 indicates refusal to release; an argument of 1 indicates that the process has released the VT. This *ioctl* is also used to indicate completion of acquiring the VT. This is done by giving an argument of VT_ACKACQ.

VT_ACTIVATE

This call has the effect of making the VT specified in the argument the active VT. The VT manager will cause a switch to occur in the same manner as if a hotkey had initiated the switch. If the specified VT is not open or does not exist the call will fail and *errno* will be set to ENXIO.

VT_WAITACTIVE

If the specified VT is already active, this call returns immediately. Otherwise, it will sleep until the specified VT becomes active, at which point it will return. The argument to this *ioctl* is a pointer to the **vt_mode** structure defined above. When this call is made, the VT manager checks to see if there is already a process with that VT. If there is not, it puts the VT in active mode. Next, the VT manager checks to see the VT passed by the arguments is already active. If so, it returns the same VT. If there are no active VTs available, it sets the VT flag to VT_WAITACT and sleeps until a VT becomes active.

VT_GETSTATE

This call is used to obtain the active VT number and a list of open VTs. The argument to this call is an address to the following structure:

```
struct vt_stat {
   ushort  v_active,  /* number of the active VT */
           v_signal,  /* signal type to be sent to open VTs */
           v_state;   /* count of openVTs.  For every 1 in this
            T   field, there is an open VT */
}
```

and a pointer to the **vt_mode** structure defined above.

In this call, the VT manager first gets the number of the active VT. Then it counts the number of open VTs in the system and sets a **1** for each open VT in **v_state**. The VT manager then transfers the information in structure **vt_stat** passed by the user process.

VT_GETSIG

This call is used to send a signal to all open VTs. The arguments to this call are a pointer to the structure **vt_mode** and a pointer to the structure **vt_stat**, both defined above. In this call, the VT manager checks whether the VT is open, and if it is, it sends the signal given in the **v_signal** field of the **vt_stat** structure.

FILES

/dev/vtxx

SEE ALSO

ioctl(2), sighold(2), signal(2), sigrelse(2), sigset(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

WARNINGS

There is a potential for a race condition on a heavily loaded system. When a process-control mode VT is sent the release requested signal, it is possible that it may not reply with a release *ioctl* before the internal timer expires and refusal to switch is assumed. The switch request will then be canceled and the VT will not switch screen faces. This can be detected by the process attempting to release the display. If the release *ioctl* fails and *errno* is EINVAL, then the releasing process can assume that the switch request was canceled.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

wt − cartridge tape device

DESCRIPTION

The *wt* device driver provides the interface to an AT cartridge (streaming) tape device. When opened for reading or writing, the tape is presumed to be positioned as desired for the operation. On close, the tape is rewound or left at the current position depending on the device selection. To rewind the tape, use the *rewind-on-close* device.

Ioctl Calls

Two *ioctl*(2) system calls are available from **<sys/wtioctl.h>**:

ERASE This call erases data on the tape cartridge.

RETENS This call is used to "retension" tape in the cartridge.

FILES

```
/dev/tape     /* rewind on close */
/dev/ntape    /* no rewind */
```

NOTES

At present the driver supports Wangtek and Bell Technologies compatible tape controller cards.

SEE ALSO

backup(1), cpio(1), dd(1), restore(1), tar(1).
dump(1) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.
IBM Personal Computer AT Technical Reference.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

xt – multiplexed tty driver for AT&T windowing terminals

DESCRIPTION

The *xt* driver provides virtual *tty(7)* circuits multiplexed onto real *tty*(7) lines. It interposes its own channel multiplexing protocol as a line discipline between the real device driver and the standard *tty*(7) line disciplines.

Virtual *tty*(7) circuits are named by character-special files of the form */dev/xt???*. File names end in three digits, where the first two represent the channel group and the last represents the virtual *tty*(7) number (0-7) of the channel group. Allocation of a new channel group is done dynamically by attempting to open a name ending in **0** with the **O_EXCL** flag set. After a successful open, the *tty*(7) file onto which the channels are to be multiplexed should be passed to *xt* via the **XTIOCLINK** *ioctl*(2) request. Afterwards, all the channels in the group will behave as normal *tty*(7) files, with data passed in packets via the real *tty*(7) line.

The *xt* driver implements the protocol described in *xtproto*(5) and in *layers*(5). Packets are formatted as described in *xtproto*(5), while the contents of packets conform to the description in *layers*(5).

There are three groups of *ioctl*(2) requests recognized by *xt*. The first group contains all the normal tty *ioctl*(2) requests described in *termio*(7), with the addition of the following:

TIOCEXCL Set exclusive use mode; no further opens are permitted until the file has been closed.

TIOCNXCL Reset exclusive use mode; further opens are once again permitted.

The second group of *ioctl*(2) requests concerns control of the windowing terminal, and is described in the header file **<sys/jioctl.h>**. The requests are as follows:

JTYPE, JMPX Both return the value **JMPX.** These are used to identify a terminal device as an *xt* channel.

JBOOT, JTERM Both generate an appropriate command packet to the windowing terminal affecting the layer associated with the file descriptor argument to *ioctl*(2). They may return the error code **EIO** if the system *clist* is empty.

JTIMO, JTIMOM **JTIMO** specifies the timeouts in seconds, and **JTIMOM** in milliseconds. Invalid except on channel 0. They may return the error code **EIO** if the system *clist* is empty.

JWINSIZE Requires the address of a *jwinsize* structure as an argument. The window sizes of the layer associated with the file descriptor argument to *ioctl*(2) are copied to the structure.

JZOMBOOT Generate a command packet to the windowing terminal to enter download mode on the channel associated with the file descriptor argument to *ioctl*(2), like **JBOOT**; but when the download is finished, make the layer a zombie (ready for debugging). It may return the error code **EIO** if the system *clist* is empty.

JAGENT Send the supplied data as a command packet to invoke a windowing terminal agent routine, and return the terminal's response to the calling process. Invalid except on the file descriptor for channel 0. See *jagent*(5). It may return the error code **EIO** if the system *clist* is empty.

The third group of *ioctl*(2) requests concerns the configuration of *xt*, and is described in the header file **<sys/xt.h>**. The requests are as follows:

XTIOCTYPE Returns the value **XTIOCTYPE.**

XTIOCLINK Requires an argument that is a structure, *xtioclm*, containing a file descriptor for the file to be multiplexed and the maximum number of channels allowed. Invalid except on channel 0. This request may return one of the following errors:

- **EINVAL** *nchans* has an illegal value.
- **ENOTTY** *fd* does not describe a real *tty*(7) device.
- **ENXIO** *linesw* is not configured with *xt*.
- **EBUSY** An **XTIOCLINK** request has already been issued for the channel group.
- **ENOMEM** There is no system memory available for allocating to the *tty*(7) structures.
- **EIO** The **JTIMOM** packet described above could not be delivered.

HXTIOCLINK Like **XTIOCLINK,** but specifies that ENCODING MODE be used.

XTIOCTRACE Requires the address of a *tbuf* structure as an argument. The structure is filled with the contents of the driver trace buffer. Tracing is enabled. This request is invalid if tracing is not configured.

XTIOCNOTRACE Tracing is disabled. This request is invalid if tracing is not configured.

XTIOCSTATS Requires an argument that is the address of an array of size S_NSTATS, of type *Stats_t*. The array is filled with the contents of the driver statistics array. This request is invalid if statistics are not configured.

XTIOCDATA Requires the address of a maximum-sized *Link* structure as an argument. The structure is filled with the contents of the driver *Link* data. This request is invalid if data extraction is not configured.

FILES

/dev/xt/??[0-7]	multiplexed special files
/usr/include/sys/jioctl.h	packet command types
/usr/include/sys/xtproto.h	channel multiplexing protocol definitions
/usr/include/sys/xt.h	driver specific definitions

SEE ALSO

layers(1), termio(7), tty(7).

ioctl(2), open(2), libwindows(3X), jagent(5), layers(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

NAME

debugger – symbolic kernel debugger

SYNOPSIS

<CTRL> <ALT> d

DESCRIPTION

The RAM-resident kernel debugger is accessed by generating an interrupt on interrupt level 1. This is frequently done by pressing a front-panel switch (press front-panel **int**). In the case of PC-type systems without front-panel interrupt switches, the debugger is activated by typing **CTRL ALT d** on the console.

The general structure of the debugger is a stack-based language, using postfix notation similar to that of the typical Reverse Polish Notation pocket calculator. Arguments are pushed on the stack, and then the operation is entered.

Pushing items onto the stack:
Items are pushed onto the stack by typing them in. Errors are printed for items that are not recognized, such as symbols that do not exist in the symbol table.

Symbols and symbolic references:
The debugger has the capability of referring to objects in the kernel by their symbolic names. Since the symbol table from the COFF image of the kernel does not get loaded at boot time, these references are resolved in a different fashion. When a kernel is built, it is processed by the utility **unixsyms** as the last step in building the kernel. This utility takes the symbols and values from the COFF image and places them in a specially reserved area of the initialized data section of the image.

Invoking operations:
Operations are invoked by entering the name of the operation (refer to "Pushing items onto the stack," above). An error will be printed if there are not enough items on the stack for the selected operation or if the selected operation doesn't exist.

Arguments may be entered on separate lines, or an entire expression may be entered on one line. For example:

200
350
+

This would leave the value **550** on top of the stack, where it can be printed, using the **p** operator, or used in future calculations. The same result can be accomplished by:

200 350 +

Arguments on the stack are referred to by their positions relative to the top of the stack. The top of the stack always contains the last item pushed. The bottom of the stack is always the first item pushed (assuming that the operation always begins with a clean stack). For example, if two items are pushed onto the stack, the latest one pushed is referred to as the <tos> (top of stack). The last item pushed is said to be on top of the stack. The first item that was pushed is referred to as <tos–1> (top of stack –1). This ordering scheme is important for operations that are not commutative or those that require different types of operators.

Arithmetic and logical operators:
The set of available arithmetic and logical operators is similar to that of the C language. The following table lists the arithmetic and logical operators. Also listed are those arguments that must be on the stack and those arguments that are left on the stack.

Operator	*No. of Operands*	*No. of Results*	*Result Description*
+	2	1	sum
–	2	1	difference (<tos–1> – <tos>)
*	2	1	product
/	2	1	integer dividend (<tos–1> / <tos>)
%	2	1	integer remainder (<tos–1> % <tos>)
<<	2	1	<tos> shifted left by <tos–1>
>>	2	1	<tos> shifted right by <tos–1>
&	2	1	result of logical *and*
|	2	1	result of logical *or*
--	1	1	decrement
++	1	1	increment
^	2	1	exclusive *or*

Relational operators:
Each of the following operators leaves a **1** on the stack if the condition is true, and leaves a **0** if it is false.

Operator	*No. of Operands*	*No. of Results*	*Result Description*
==	2	1	1 if <tos–1> and <tos> are equal
!=	2	1	1 if <tos–1> and <tos> are not equal
>	2	1	1 if <tos–1> is greater than <tos>
<	2	1	1 if <tos–1> is less than <tos>
||	2	1	1 if either <tos–1> or <tos> is non-zero
&&	2	1	1 if both <tos–1> and <tos> are non-zero

Debugger control operators:
The following operators control the behavior of the debugger itself. Most of them are related to the user interface.

Operator	*No. of Operands*	*No. of Results*	*Result Description*
verbose	0	0	places the debugger in verbose mode useful for debugging the debugger, little else
nonverbose	0	0	takes the debugger out of verbose mode
ibase	1	0	sets the debugger input base to <tos>
ioctal	0	0	sets the debugger input base to **8** (octal)
idecimal	0	0	sets the debugger input base to **10** (decimal)
ihex	0		sets the debugger input base to **16** (hex)
ibinary	0	0	sets the debugger input base to **2** (binary)
ooctal	0	0	sets the debugger output base to **8** (octal)
odecimal	0	0	sets the debugger output base to **10** (decimal)
ohex	0	0	sets the debugger output base to **16** (hex)
stk	0	0	displays the debugger operation stack
clrstk	X	0	clears the debugger operation stack
sysdump	0	0	causes a memory image to be written on **swapdev**

Reading and writing memory, I/O ports, and registers:
In order to read the contents of a memory location, I/O, or register, first push the address of the memory location or I/O port on the stack. In order to read a register, it is necessary to push a non-printable token, which refers to the register in question, onto the stack. Memory and I/O address can be entered either as numbers in the current input base or as symbolics. Registers are referred to by a percent sign (%), followed by the register name (e.g., *%eax*). The operators to read memory locations and registers are **r1**, **r2**, and **r4**. These operators read 1, 2, or 4 bytes respectively. The value read is left on the top of the stack. In order to write a register or memory location, the value to be written must be pushed first, then the address or register name to which it is to be written. The operators for writing are **w1**, **w2**, and **w4**. The same syntax applies to reading and writing I/O ports. The operators for reading and writing I/O ports are **rio1**, **rio2**, **rio4**, **wio1**, **wio2**, and **wio4**. The following table summarizes the *read* and *write* operators.

Operator	*No. of Operands*	*No. of Results*	*Result Description*
r1	1	1	the byte read from address <tos>
r2	1	1	the word read from address <tos>
r4	1	1	the long word read from address <tos>
w1	2	0	the byte value <tos–1> is written to address <tos>
w2	2	0	the word value <tos–1> is written to address <tos>
w4	2	0	the long word value <tos–1> is written to address <tos>
rio1	1	1	the byte read from port <tos>
rio2	1	1	the word read from port <tos>
rio4	1	1	the long word read from port <tos>
wio1	2	1	the byte value <tos–1> is written to I/O port <tos>
wio2	2	1	the word value <tos–1> is written to I/O port <tos>
wio4	2	1	the long word value <tos–1> is written to I/O port <tos>

Special commands:
Among the special commands are commands to print **inode** structures, process table entries, and disassembled 80386 assembly code.

To display disassembled code, the starting address from which to disassemble must be pushed, followed by the number of lines of assembly code to print (remember this is in the input base). The operator is **dis.**

Process table entries can be displayed by invoking the operator **ps**. This operator takes no arguments and leaves no results on the stack.

The **sleeping** operator displays the process IDs and **WCHAN** values of all processes that are sleeping. Like **ps**, **sleeping** requires no arguments on the stack and leaves no results there.

The **stack** operator displays a symbolic stack backtrace from the current point of execution. It needs no arguments and leaves no result on the stack.

The **findsym** operator takes one argument on the stack, an address, and displays the symbol with the highest address which is equal to or below the argument address. This allows the user to find the name of the routine or data structure into which a given address is pointing.

The **dump** operator displays a specified number of words of memory, starting at a specified location. The starting location is pushed on the stack first, followed by the count of words to display. Words are displayed grouped as bytes, and are displayed from top to bottom, left to right, *higher address locations first.* In this way, byte and word ordering is always correct.

On the right-hand side of the display are the ASCII equivalents of the bytes displayed on the left-hand side, which are displayed in hex. The output of the **dump** operator is unaffected by the current output base of the debugger; it is always displayed in hex.

The **stackdump** operator displays the entire kernel stack and user structure. This dump is printed in the same format as that of the dump command (see above). The **pinode** operator prints information contained in an **inode** structure. It takes one argument on the stack, which must be the address of an **inode** structure. Information printed includes the address of the **inode** (relative to the kernel symbol **inode**), the reference count, the device number, the **inode** number, the number of links to this **inode**, the owner and group IDs of this **inode**, and the size of the file referred to by this **inode**.

The **gdevcblk** command takes one argument, an address, and prints in English all structure items of the **gdev_ctl_block** structure with items indicated where the use of other commands is appropriate.

The **gdevpblk** command takes one argument, an address, and prints in English all structure items of the **gdev_parm_block** structure with items indicated where the use of other commands is appropriate.

The **gdevpart** command takes one argument, an address, and prints information on the partition table of the **gdev_parm_block** structure.

The **gdevdrv** command takes one argument, an address, and prints information on the **gdev_driver** of the **gdev_parm_block** structure.

The **print_chain** command takes one argument, an address, and prints the **drq** chain for a **gdev_parm_block** item.

The **print_proc** command takes one argument, an address, and prints the **proc** structure information in English.

Breakpoints:
The debug registers on the chip are available directly to the debugger. In addition, there are some commands for setting breakpoints and single-stepping conveniently. The debug registers are read and written with the following operators:

Operator	*No. of Operands*	*No. of Results*	*Result Description*
db0	0	1	leaves the contents of debug register 0 on stack
db1	0	1	leaves the contents of debug register 1 on stack
db2	0	1	leaves the contents of debug register 2 on stack
db3	0	1	leaves the contents of debug register 3 on stack
db6	0	1	leaves the contents of debug register 6 on stack
db7	0	1	leaves the contents of debug register 7 on stack
wdb0	1	0	sets db0 to <tos−1>
wdb1	1	0	sets db1 to <tos−1>
wdb2	1	0	sets db2 to <tos−1>
wdb3	1	0	sets db3 to <tos−1>
wdb6	1	0	sets db6 to <tos−1>
wdb7	1	0	sets db7 to <tos−1>

The functions of each of the debug registers can be found in the Intel *80386 Programmer's Reference Manual* (Order No. 230985−001).

There are also explicit operators to set breakpoints. These operators are **brk0, brk1**, **brk2**, **brk3** ... These operators all have the same syntax; the only difference is that each one uses a different debug address register. These operators need two arguments on the stack when executed. The first of these arguments is the address or symbol at which to set a breakpoint. The second argument is a token specifying which type of breakpoint to set. The possible breakpoint type tokens are:

Token	*Breakpoint Type*
.i	break on instruction execution
.a	break on byte access (*read/write*)
.aw	break on word access (*read/write*)
.al	break on long word access (*read/write*)
.m	break on byte modify (*write*)
.mw	break on word modify (*write*)
.ml	break on long word modify (*write*)
.clr	clear the breakpoint

The **db**? operator displays the current status of the debug registers.

For example, to set a breakpoint at the routine *namei* in the kernel, type:

namei .i brk0

This sets an instruction execution breakpoint at the first instruction in the routine *namei*. As another example, assume you would also like to break into the debugger whenever a *write* takes place to the data location *swapdev*. Further, suppose *swapdev* is a word (16-bit) object. To set this breakpoint, enter:

swapdev .mw brk1

Notice the use of break register **1** this time; this is to avoid using the break register used by the instruction breakpoint above.

To clear a breakpoint, for example, the breakpoint set at *namei* above, enter:

0 .clr brk0

In addition to allowing the setting of breakpoints using the on-chip debug facilities, the debugger also has the capability to single step instructions. This is done by typing **s**. Subsequent single steps may be performed by using **RETURN.**

SEE ALSO

crash(1M).

BUGS

These are the currently known bugs. They are being investigated.

The **sysdump** command doesn't always work.

Due to variable initialization problems, after entering the debugger, always execute the **stack** command first.

The debugger uses tab characters to separate columns of output, so make sure tabs are set on the terminal before entering the debugger.

The **ps** operator is not implemented at this time.

After breaking into the debugger at an instruction breakpoint, register access operators access the register set from the user context, not the kernel.

The **dis** function sometimes enters an endless loop when given too large a **count**. If this occurs, use **CTRL ALT d** to re-enter the debugger.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

lcasep – convert first field to lowercase

SYNOPSIS

lcasep [**–f** infile] [**–o** outfile]

DESCRIPTION

The *lcasep* program converts all uppercase characters in the first field of each input line to lowercase and writes the line to its output. By default, *lcasep* reads from the standard input and writes to the standard output. Fields are delimited by a tab (*ascii 0x9*) character. It is used in preparation for sorting *smail*'s *paths* database.

SEE ALSO

pathproc(8), smail(8).
paths(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.
pathalias by Peter Honeyman.

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

NAME

mkdosfs – construct a DOS file system in a UNIX System environment

SYNTAX

/etc/mkdosfs *device_name* **blocks** [*:rootdirs*] [*cluster_size*] [**−p**]
/etc/mkdosfs *device_name* **proto** [*cluster_size*] [**−p**]

DESCRIPTION

The *mkdosfs* program constructs a DOS file system on a UNIX System by writing on the *device_name*. The *device_name* may be a block special device (diskette or fixed disk) or a UNIX System file. The number of *blocks* specifies the number of (512 byte) sectors in the file system. *mkdosfs* builds a file system with a **root** directory and includes the prototype files if *proto* is found on the command line.

The optional arguments allow for customization of the file system to fit the needs of the user. *rootdirs* specifies the number of **root** directory entries (the default is 128 on a fixed disk and follows DOS conventions on diskettes). *cluster_size* specifies the number of (512 byte) sectors in a cluster (the default is 1 on a fixed disk and follows DOS conventions on diskettes).

If **−p** is specified, the IBM PC-DOS algorithm for determining the size of a DOS FAT (File Allocation Table) entry will be used instead of the default MS-DOS algorithm. This only applies to DOS file systems created on fixed disks. VP/ix users should use the default (no **−p**) algorithm.

If *device_name* is a UNIX System file, the DOS file system created in that file will be suitable for later use on fixed disks and RAM disks. It will *not* work on a diskette.

When using *mkdosfs* on a diskette, the diskette must already be formatted.

The following examples create DOS file systems on diskettes that will be compatible with MS-DOS and PC-DOS systems:

```
360K:  mkdosfs /dev/dsk/f0d9dt 720
1.2MB:  mkdosfs /dev/dsk/f0q15dt 2400
```

SEE ALSO

mkfs(1M).

ADDED VALUE

This entry, supplied by INTERACTIVE Systems Corporation, is an extension of UNIX System V.

FILES

Except for **/usr/lib/sendmail.cf**, these path names are all specified in **/usr/lib/sendmail.cf**. Thus, these values are only approximations.

/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.pag /usr/lib/aliases.dir	database of alias names
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	help file
/usr/lib/sendmail.st	collected statistics
/usr/spool/mqueue/*	temp files

SEE ALSO

mail(1), mailx(1), rmail(1).
syslog(3), aliases(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.
DARPA Internet Request For Comments RFC819, RFC821, RFC822, "Sendmail – An Internetwork Mail Router" and "Sendmail Installation and Operation Guide" in the *INTERACTIVE UNIX Operating System Guide*.

NAME

smail, rmail – UUCP mailer with routing

SYNOPSIS

smail [options] address ...
rmail [options] address ...

DESCRIPTION

The *smail/rmail* program replaces **/bin/rmail** (*rmail*(1)) to become the UUCP mail transport mechanism. They are links to the same executable. *rmail* receives mail from UUCP; *smail* introduces mail into UUCP.

smail/rmail can work with or without *sendmail*(8) or another intelligent mail system. For hosts with only **/bin/mail** (*mail*(1)), *smail/rmail* subsumes some of the functions of *sendmail*, and hands only local mail to **/bin/mail**. For hosts with *sendmail*, *smail/rmail* can act as UUCP frontends and backends to *sendmail*, allowing *sendmail* to process all mail through the host. As distributed, *bang* mail that is not bound for a local recipient will be passed directly to *uux* without calling *sendmail*.

To varying degrees, *smail/rmail* automatically routes the addresses it processes. *smail/rmail* most often routes domain-style addresses (i.e., *user@domain*), producing a UUCP path (i.e., *host!address*) or a local address (i.e., *user*), but it can also reroute explicit UUCP paths.

Options

–A Print the resolved addresses. Do not collect a message or invoke a mailer.

–d Be verbose and do not invoke other mailers.

–v Be verbose, but still invoke other mailers.

–h *hostname*
Set *hostname*. The default is configuration dependent, but it is usually provided by a system call such as *gethostname*(2) or *uname*(2).

–H *hostdomain*
Set *hostdomain*. The default is configuration dependent.

–F *address*
Use *address* on the "From:" line in locally generated mail.

–p *pathfile*
Set path database file name if it is not **/usr/lib/uucp/paths**.

–a *aliasfile*
For sites without *sendmail*, set alias database file name if not in the place defined at compile time (see ALIAS in **/etc/default/smail**). This is usually **/usr/lib/aliases**.

–n *namelist*
smail supports another type of aliasing intended for full name resolution using a sorted file called *namelist* which contains name/address pairs. This allows mail to *George.P.Burdell@gatech.edu* to be delivered correctly. These aliases are very simple since they are not composed of long lists of recipients for each alias. They are also numerous,

because mail to *George.P.Burdell* may be addressed to *Burdell*, *G.Burdell*, *George.Burdell*, *P.Burdell*, *G.P.Burdell*, or *George.P.Burdell*. This simpler form of aliasing uses the same fast searching algorithm that is used for the *paths* file, so it keeps resolution time manageable.

−q *number*
Take *number* as the queueing threshold. When routing mail (**−r**, **−R**, or domain addressed mail) to a given host, if the cost listed in the *paths* file is less than the queueing threshold, then the mail will be sent immediately. This overrides the default threshold (see QUEUECOST in **/etc/default/smail**) of DEDICATED+LOW.

−m *number*
At most, *number* jobs will be handed to *uux* for immediate delivery by a single invocation of *smail* (see MAXNOQUEUE in **/etc/default/smail**).

−u *uuxflags*
Use *uuxflags* as the flags passed to *uux* for remote mail. This overrides any of the default values and other queueing strategies.

-c Consult the *paths* file for the cost of the path even when not routing the mail. This makes it possible to use the cost information when sending pure UUCP path mail without rerouting it.

−r Route the first component of a UUCP path (*host!address*) in addition to routing domain addresses (*user@domain*).

−R Reroute UUCP paths, trying successively larger right-hand substrings of a path until a component is recognized.

−l Instead of routing a domain address, send it to the local mailer for processing. Normally, only local addresses go to the local mailer.

−L Send all addresses to the local mailer for processing, including UUCP paths.

Most of the flags are also compile time options, since *uux* does not normally invoke *rmail* with the desired flags. *smail* resets any preset **−l** or **−L** flags. The **−l** flag causes **rmail** to send all domain addresses through the local mailer, to process addresses for non-UUCP domains. The **−L** flag causes *rmail* to send even explicit UUCP paths through the local mailer, presumably to make use of other transport mechanisms. In both cases, *rmail* defers any routing until *smail* assumes control of it.

Configuration

The configuration of the smail program can be changed by editing the file **/etc/default/smail**. Remove the comment character (#) in front of the option that you want to change and enter the new value after the equal sign (=). The configurable options and their defaults are listed below.

HOSTNAME
The literal name of this host. If defined, **HOSTNAME** is used instead of calling *uname()* to determine the local hostname.

MYDOM *.uucp*
The literal domain suffix for this host. The suffix *.uucp* here is used only for testing; you should use a site name that has been registered with COM, EDU, etc.

HOSTDOMAIN
Defining this overrides the default **HOSTNAME.MYDOM.**

DOMGATE *no*
Act as a Domain Gateway. If defined, **DOMGATE** (DOMain GATEway) will cause addresses of the form:

user@MYDOM or MYDOM!user

(with and without the leading dot (.) on MYDOM) to be treated simply as *user* – a purely local address. Then, it is left to the aliasing code to map it back to a non-local address if necessary.

HIDDENHOSTS *no*
Hide subdomains of hostdomain. Allows hosts that serve as domain gateways to hide the subdomains beneath them. Mail that originates at any of the hosts in the subdomain will appear to come from the gateway host. Thus, mail from

anything.hostdomain!user

will appear to come from

hostdomain!user

A consequence of this is that return mail to *hostdomain!user* would need to be forwarded to the proper subdomain via aliases or other forwarding facilities. If you are using *sendmail*, if **HIDDENHOSTS** is defined here, it should be used in ruleset 4 of the *sendmail.cf*, too.

SMARTHOST *smart-host*
pathalias alias for relay host. Mail that would otherwise be undeliverable will be passed to the aliased **SMARTHOST** for potential delivery. It is important to receive prior approval to use the host you specify in your *pathalias* input as a relay. If you are using *foovax* as your relay, and you define **SMARTHOST** as *smart-host*, then the *pathalias* alias would be:

smart-host = foovax

FULLNAME
Full name and address pairs. Defining **FULLNAME** means that Full Name resolution will be attempted when necessary. The Full Name information will be taken from a list of {Full Name, address} pairs. The names in the list must be sorted without regard to uppercase/lowercase; all full name searches are case insensitive.

PATHS */usr/lib/uucp/paths*
The name of the *pathalias* database file.

LOG Log file of UUCP mail addresses. Define **LOG** if you want a log of mail. This can be handy for debugging and traffic analysis.

RECORD
Record file of UUCP mail. Define **RECORD** in order to get a copy of all mail sent. This uses a great deal of time and space, so it is only used for extreme debugging cases.

DEFQUEUE *yes*
Queue *uux* mail instead of sending it immediately.

QUEUECOST *100*
Remote mail with a cost of less than **QUEUECOST** will be handed to *uux* for immediate delivery.

Addresses

smail/rmail understands *user@domain* to be a domain address, *host!address* to be a UUCP path, and anything else to be a local address.

Because some versions of *rmail* unpredictably interpret mixed UUCP/domain addresses, *smail/rmail* understands *domain!user* to be a domain address, and generates *path!domain!user* when mailing to a cognate *smail/rmail* host. To distinguish domain *domain!user* from UUCP *host!address*, *domain* contains at least one (1) dot. Unlike the old versions of **/bin/rmail**, *smail/rmail* gives precedence to @ over ! when parsing mixed addresses. Thus, *a!b@c* is parsed as (*a!b*)@*c*, rather than *a!*(*b@c*).

Routing

Because *smail/rmail* is the UUCP transport mechanism, it can only effect delivery on UUCP paths and local addresses; domain addresses require resolution into UUCP paths or local addresses. To resolve a domain address, *smail/rmail* finds a route to the most specific part of the domain specification listed in the routing table. Two degrees of resolution can occur:

Full resolution: *smail/rmail* finds a route for the entire domain specification, and tacks the user specification onto the end of the UUCP path. The address can also fully resolve to a local address (the UUCP path is null).

Partial resolution:
smail/rmail finds a route for only the right-hand part of the domain specification, so it tacks the complete address (in the form *domain!user*) onto the end of the UUCP path. Since this syntax is not widely understood, UUCP gateways listed in the path database must install new UUCP software, either *smail/rmail* or new *sendmail* configuration files (or both).

It is an error if a partially resolved address routes to the local host (a null UUCP path), since according to the routing table, the local host is responsible for resolving the address more fully.

The **−r** flag causes *smail/rmail* to attempt to route only the first component of a UUCP path. If this fails, it passes the unrouted address to *uux,* in case the path database is not complete. The **−R** flag causes *smail/rmail* to take a UUCP path and route the rightmost component of the path (save the user name) possible. This is mostly for hosts that have very up-to-date routing tables.

If a route cannot be discerned from the available routing database, then one more attempt to route the mail is made by searching for an entry in the database for a route to a *smart-host*. If this entry exists, then the mail will be forwarded along that route to be delivered. This allows a host to depend on another (presumably better informed) host for delivering its mail. This kind of arrangement should be made *in advance* with the *smart-host*'s system administrator.

After *smail/rmail* resolves an address, it reparses it to see if it is now a UUCP path or local address. If the new address turns out to be another domain address, *smail* complains because a domain name should always resolve to either a UUCP path or the local host.

By default, *smail* will not alter the explicit *bang* path routing of any mail message. If the stated path is unusable (i.e., the next hop host is unknown), then *smail* will apply ALWAYS routing, and attempt to deliver the mail to the potentially new address. If this fails too, then REROUTE routing will be applied to the address, and another attempt to deliver is made. Lastly, an attempt to find a path to a better informed host *smart-host* will be made and the mail passed to that host.

From Lines

smail/rmail collapses "From" and ">From" lines to generate a simple from argument which it can pass to *sendmail* or use to create its own "From" line. The rule for "fromming" is concatenate each "remote from" host (separating them by !'s), and tack on the address on the last "From" line; if that address is in *user@domain* format, rewrite it as *domain!user*; ignore host or domain if either is simply the local *hostname*. It also removes redundant information from the "From" line. For instance:

...!myhost!myhost.mydomain!...

becomes:

...!myhost!...

Leading occurrences of the local host name are elided as well.

smail/rmail generates it own "From" line, unless it is feeding *sendmail*, which accepts the **−f***from* argument. For UUCP bound mail, *smail/rmail* generates a "remote from hostname," where hostname is the UUCP hostname (not the domain name), so that "From" can indicate a valid UUCP path, leaving the sender's domain address in "From:."

Headers

Certain headers ("To:," "From:," "Date," etc.) are required by RFC822. If these headers are absent in locally generated mail, they will be inserted by *smail*. Also, a line of trace information, called a "Received:" line, will be inserted at the top of each message.

Undeliverable Mail"

Although nobody likes to have a mail message fail to reach its intended destination, it sometimes happens that way. Mail that is found to be undeliverable (i.e., unknown user or unknown host) will be returned to the sender.

FILES

/usr/lib/uucp/paths	ASCII path database
/usr/lib/aliases	ASCII alias database
/usr/spool/uucp/mail.log	log of mail
/tmp/mail.log	record of mail

SEE ALSO

mail(1), uucp(1), uux(1C), sendmail(8).
aliases(5), paths(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NOTES

INTERACTIVE

A Kodak Company

DOC0109-2Z