A/UX Command Reference was written, edited, and composed on a desktop publishing system using Apple Macintosh computers, and `troff` running on A/UX. Page proofs were created on Apple LaserWriter printers. Final pages were output directly to 70 millimeter film on an Electrocomp 2000 Electron Beam Recorder. PostScript, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type and display type are Times, Garamond, and Helvetica. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier, a fixed-width font.

Writers: Erik Akin, Mike Elola, Kristi Fredrickson, Michael Hinkson, Linda Kinnier, Paul Pannish, Cheryl Salgado, Kathy Wallace, and Laura Wirth

Writing Group Lead: Mike Elola

Developmental Editor: Silvio Orsino

Art Director: Tamara Whiteside

Production Editor: Jeannette Allen

Production Supervisor: Robin Kerns

Special thanks to Anne Szabla and Chris Wozniak

A/UX Command Reference

Sections 1(Q–Z) and 6

Release 3.0

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

A/UX Command Reference

Contents

About This Manual

This manual is one of three primary manuals in the set of A/UX reference manuals. *A/UX Command Reference*, *A/UX Programmer's Reference*, and *A/UX System Administrator's Reference* contain information about most of the provisions of A/UX, such as its commands, its library routines, its system calls, and its file formats.

These reference manuals constitute a compact encyclopedia of A/UX information. As in an encyclopedia, the information is subdivided into subdocuments, or ''manual pages.'' The information in each manual-page subdocument adheres to a distinctive presentation format. For example, information about command syntax is consistently presented under the heading ''Synopsis.'' (This format is described in detail later in this preface.)

Because most of us need occasional reminders regarding the order and kind of arguments that can accompany a command, the information in the ''Synopsis'' and ''Arguments'' sections is presented for use by users at all levels. However, the information in the ''Description'' section is often written for more advanced users; novices most likely will not be able to learn about the provisions of A/UX from these reference manuals alone.

Because these reference manuals are not intended to be tutorials or learning guides, they should not be the first A/UX books you read. If you are new to A/UX or are unfamiliar with a specific functional area (such as the Macintosh Finder), you should first read *A/UX Essentials* and the other A/UX user guides. After you have worked with A/UX, the reference manuals can help you understand new features or refresh your memory about features you already know.

Manual pages: a standard for presenting information

The headings conventionally used in the manual pages have virtually become an industry standard for reference documents. Furthermore, the way that this large collection of subdocuments is conventionally organized into sections and books is also something of a standard.

Despite the standardization, locating specific information within this large body of documentation can often be difficult. First you must locate the correct manual page. Once you have the correct manual page, you can usually go directly to the correct subsection.

To help you locate information, you should read the next section, which explains several means of finding the information you need.

To help you learn to use these books more effectively, other sections in this preface describe the presentation standards that are being used. Some of these are organizational standards that apply at the book and section level. Other conventions and content standards apply within the scope of each manual page, such as the use of standard subheadings and the conventional use of certain fonts and text styles.

Note that the most durable standards have been the standards that apply to the organization and primary headings of each manual page. Of course there are areas in which the A/UX reference books are exceptional, particularly in their more regular use of headings. These books also deviate from industry standards in a few typographic and style areas, which are described later in this preface. For example, the Courier font is used consistently to represent text that is displayed in a terminal window or entered as part of a command line. Other UNIX® books often use boldface type to represent such text.

There has been more instability with respect to how the manual pages are collected into sections and books. For more detailed information, see ''Previous Organization of Sections into Books'' later in this preface.

Locating information in the reference manuals

You can locate information in the reference manuals by using one of the following tools:

- Table of contents. Each reference manual contains one general table of contents for the entire manual. Located at the beginning of each new section of manual pages is a detailed table of contents. (If a section must span from one binder to another, a tailored table of contents is provided for each of the subdivisions.) The general table of contents lists the sections covered in the complete manual. The detailed table of contents lists the manual pages contained within one section (or section subdivision) along with a brief description of the A/UX provision that is covered in each manual page.

- Query commands. The `man`, `whatis`, and `apropos` commands display on-screen all the information contained in a manual page or just the information in the ''Name'' section of one or more manual pages that satisfy a search criterion. The next sections tells you how to use the on-line versions of the manual pages.

- *A/UX Reference Summary and Index.* This separate manual is considered part of the A/UX set of reference manuals, but it is not a ''standard'' resource like the other reference materials. Its primary purpose is to help you locate the correct manual page to refer to in other books. From its summaries, you might also occasionally find all the information you required. It contains the following subsections:
  - ''Commands by function.'' This subsection classifies the A/UX user and system administrator commands by the general or most important function each performs. The summary gives you a broader view of the commands that are available and the context in which each is often used. Each command is mentioned just once in this listing.
  - ''Command synopses.'' This subsection is a compact collection of syntax descriptions for all of the commands in *A/UX Command Reference* and in *A/UX System Administrator's Reference.* It may help you find the syntax of commands more quickly when the syntax is all you need.
  - ''Index.'' The index lists key terms associated with A/UX subroutines and commands. These key terms can help you locate the manual page you need when you don't know if such a keyword-related command or subroutine exists.

The index provided in *A/UX Reference Summary and Index* is designed to be more compact and easier to use than the more industry-standard permuted index, which indiscriminately indexes manual pages under each of the words found in their ''Name'' sections.

The manual pages listed in the index portion *A/UX Reference Summary and Index* are indexed under more than one entry; for example, `lorder`(1) is included under ''archive files,'' ''sorting,'' and ''cross-references.'' By using this type of index, you are more likely to find the reference you are looking for on the first try.

Using the on-line documentation

In addition to the paper documentation in the reference manuals, A/UX provides several ways to search and read the contents of each manual page from your A/UX system. An advantage to the on-line version of the documentation is that the computer performs the work of filtering out (or skipping) all the manual pages other than the one you specifically queried. The only prerequisite is that you already know its name (or a proper search string). However, you don't have to know how manual pages are organized by section numbers and by book titles.

To display a manual page on your screen, enter the `man` command followed by the name of the manual page you want to see. For example, to display the manual page for the `cat` command, including its description, syntax, options, and other pertinent information, you would enter

```
man cat
```

After the first screen of the text of a manual page appears, you can display subsequent screens of the text with each press of the SPACE BAR, until you reach the end of the man page. To display subsequent text one line at a time, press RETURN instead of the SPACE BAR. By pressing Q, you can quit the `man` command before viewing all of the manual page.

To display the descriptive information in the ''Name'' section of any manual page, enter the `whatis` command followed by the name of the provision you want described. In the following example, the command prompt is the percent sign, and the provision that is being queried is the `ls` command:

```
% whatis ls
ls(1)            - lists the contents of a directory
% █
```

To display a list of all manual pages whose ''Name'' sections contain a given keyword or string, enter the `apropos` command followed by a search word or search string enclosed in double quote characters. The names of A/UX provisions are listed on separate lines along with the descriptive information in the ''Name'' section of the manual page that describes those provisions. Sometimes several A/UX provisions are listed on the same line. In those cases, several A/UX provisions are described on a single manual page. You can tell which of these names is the formal name for the manual page because it will be followed by parentheses and an enclosed section number. In the following example, the command prompt is the percent sign, and the A/UX provisions that are queried are those which are described in manual pages whose ''Name'' section contains the word ''tape'':

```
% apropos tape
mt(1)           - magnetic tape manipulating program
frec(1M)        - recover files from a backup tape
mtio(7)         - interface conventions for magnetic tape devices
tc(7)           - Apple Tape Backup 40SC device driver
% █
```

These documentation query commands are described more fully in the manual pages `man`(1), `whatis`(1), and `apropos`(1) in *A/UX Command Reference.*

Book- and section-level presentation standards

Customarily, three books are used to house three collections of manual pages that are of concern to three different audiences:

- *A/UX Command Reference* is intended for users with normal file and device access privileges.
- *A/UX System Administrator's Reference* is intended for system administrators or equivalent users with unlimited device and file access privileges.
- *A/UX Programmer's Reference* is intended for programmers.

These books are further divided into sections, each of which contains a set of manual pages in alphabetical order. The standard sections and the audiences they serve are as follows:

- For users with normal access privileges, Section 1 and Section 6 describe utility and game commands.
- For users with unlimited access privileges, Section 1M and Section 8 describe system maintenance commands.
- For programmers, Section 2 describes system calls, Section 3 describes library routines, Section 4 describes file formats, Section 5 describes miscellaneous A/UX provisions, and Section 7 describes drivers and interfaces for devices.

While most of the manual pages describe an A/UX provision of some sort, there is one important exception per section: The first manual page in Sections 1, 1M, 2, 3, 4, 5, 6, 7 and 8 has the same name, `intro`. The `intro` manual pages do not describe a command or other provision of A/UX. Instead, they serve as an introduction to the rest of the manual pages in the section, providing section-specific information and conventions. (These section-introduction manual pages are also exceptions in terms of the normal alphabetical arrangement of manual pages inside sections.)

For example, the manual page `intro`(2) introduces you to return values and provides an exhaustive list of error code values and their associated error strings. In the rest of the Section 2 manual pages, the error codes are mentioned briefly or merely listed, without detailed explanations.

More advanced readers will probably have occasion to use more than one of the reference manuals. For example, manual pages in the *A/UX Programmer's Reference* frequently make references to manual pages in sections contained in the other two primary reference manuals.

More information about the organization of the reference books is given later in this preface in ''Current Organization of Sections into Books.''

How manual-page information is presented

The name of the manual page normally appears in both upper corners of each physical page. Some manual pages describe several routines or commands. For example, `chown` and `chgrp` are both described in a manual page with the primary name `chown`(1) at the upper corners. If you turn to the page `chgrp`(1), you find a reference to `chown`(1). (These cross-reference pages are included only in *A/UX Command Reference* and *A/UX System Administrator's Reference*.) However, if you enter the command `man chgrp`, the extended-coverage `chown`(1) manual page is displayed automatically.

All of the manual pages have a common format that uses the following subheadings. For the most part, the same kind of information appears under each of these subheadings. However, for manual pages that describe different kinds of A/UX provisions, the information under the same heading may differ. So, for example, the heading ''Synopsis'' contains syntax illustrations for Sections 1, 1M, and 8, but contains C declaration statements for Sections 2 and 3.

NAME
This section lists the names of the commands, programming routines, or other A/UX provisions that are described in the manual page. A succinct statement of their purpose is also provided.

SYNOPSIS
This section provides the syntax of a command or the data-type declarations associated with a programming routine.

ARGUMENTS
This section lists and describes the command options and arguments that can follow the command name on the command line.

DESCRIPTION
This section describes in detail the usage of a particular command or programming provision.

EXAMPLES
This section offers representative command lines that illustrate various uses of a command.

STATUS MESSAGES AND VALUES
This section describes possible error outcomes and, when applicable, possible success outcomes. For commands, exit values are not usually described if the command produces the customary zero exit value for success and a nonzero exit value for failure. For programming routines, the return value from a function is often an indication of completion status. In such cases, the return value is normally discussed in the ''Description'' section as well as in this section.

WARNINGS
This section describes possible usage scenarios that can damage the file system or file integrity or that produce results you would not normally anticipate.

LIMITATIONS
This section describes how the performance of a command or routine could become unreliable, or areas of functionality that an A/UX provision does not address.

NOTES
This section provides miscellaneous information regarding a command or routine, such as author or copyright information.

FILES
This section lists any files needed by the command, along with a brief description that identifies it as a file, directory, or link.

SEE ALSO
This section provides a list of references to related information.

Visual conventions for the A/UX reference manuals

A/UX books follow specific styling conventions. For example, words that require special emphasis appear in specific fonts or styles. This section describes

Term	Action
Click	Press and then immediately release the mouse button.
Choose	Activate a command that appears in a menu. To choose a command from a pull-down menu, position the pointer on the menu title and, while holding down the mouse button, slide the mouse toward you until the command is highlighted. Then release the mouse button.
Drag	Position the pointer on an icon, press and hold down the mouse button while moving the mouse so that the icon moves to the desired position, and then release the mouse button.
Enter	Type the series of characters indicated, then press the RETURN key.
Press	Press one key only. (Do not press the RETURN key afterward.)
Select	To select an icon, position the mouse pointer on the item, then click (see ''Click,'' above). To select text, use a drag-style operation (see ''Drag,'' above). When selecting a range of text, the drag operation highlights the text from the starting point over and across lines to the final position of the pointer when the mouse button was released.
Type	Type the series of characters indicated, without pressing the RETURN key afterward.

The Courier font

Throughout the A/UX reference manuals, words that appear on the screen or that you must type exactly as shown are in the Courier font.

An ''argument'' is any element that follows the command name. Command arguments other than command options typically specify the objects upon which the command should act. You often supply the names of files that you want a command to process, so *file* is frequently the last element in syntax illustrations.

Brackets and ellipsis characters in a syntax illustration should be considered part of a syntax notation. This is represented by the use of body font instead of Courier for these characters. Their font treatment tells you that you are not supposed to type these characters as part of the command line. Their meaning as a syntax notation is described next.

The brackets enclose an optional item or a group of optional items. If an optional item has constituent parts that are also optional, these parts are themselves enclosed in brackets, as in this syntax illustration:

`lpr [-i` [*numcols*]]

This syntax illustration shows that the indent (`-i`) command option can be followed by the number of columns to indent the printed page. It also shows that you can omit the number of columns; if you do, the `lpr` command uses the default indent value.

An ellipsis (...) follows an argument that can be repeated any number of times on a command line. If the ellipsis follows a bracketed group of items, the group of items can be repeated any number of times on the command line.

When command options are mutually exclusive, they cannot both be specified at the same time. In such cases, more than one syntax illustration is usually provided:

`pax -r`[*other-option-for-archive-reading*]...
`pax -w`[*other-option-for-archive-writing*]...

Outside of syntax illustrations, command options are shown with a leading hyphen also in the Courier font. When you supply multiple command options in an actual command line, only one leading hyphen is normally required. For example the following command line contains two options, `-r` and `-f`:

```
pax -rf /dev/rfloppy0
```

In the example, the `-f` option (pronounced ''minus f'') is entered without its own hyphen, even though when mentioned in running text it appears with a leading hyphen.

Because the original section numbers were preserved and then sections were recollated in accordance with the audience they served, the resulting books do not, for the most part, contain sequentially numbered sections.

The next section explains in detail how the sections are currently mapped into books.

There was another factor that led to the need to preserve the original section numbers. Some routines, system calls, and commands have the same names. To allow you to distinguish one from another, the section number is often included along with the name. While new section numbers could have helped distinguish these entities, the old numbers were much more familiar to UNIX users.

Besides distinguishing amongst identically named A/UX provisions, the section number helps identify each manual page as one that describes a command, a system call, a library routine, and so forth. Regular UNIX users sooner or later memorize what category is identified by each section number. After doing so, you can deduce how the sections must be split up into books—since each book serves a particular audience and each section category also goes along with a particular audience, the match-ups become fairly easy for you to make. The memorization part of this task is more or less considered an initiation rite for those who wish to learn to use UNIX effectively.

Until the 3.0 release of A/UX, the organization of sections into books was static. With the 3.0 release however, Section 7 has been moved out of *A/UX System Administrator's Reference* and into *A/UX Programmer's Reference.* This means that command provisions are now the exclusive focus of both *A/UX Command Reference* and *A/UX System Administrator's Reference.*

Current organization of sections into books

All manual pages are grouped by section. The sections are grouped by general function and are numbered according to standard conventions as follows:

1	User Commands
1M	System Maintenance Commands
2	System Calls
3	Subroutines
4	File Formats

5 Miscellaneous Facilities

6 Games

7 Drivers and Interfaces for Devices

8 A/UX Startup Shell Commands

Each group or section of manual pages is located in one of the reference books. Each reference book may comprise more than one binder. This section explains where these sections are currently located with respect to the three primary reference books. It also describes any subcategories that may be present in a given section.

A/UX Command Reference contains Sections 1 and 6.

- Section 1—User Commands
  This section describes commands that require no special access privileges. The commands in Section 1 may also belong to a special category, such as networking commands. Where applicable, these categories are indicated by a letter designation that follows the section number. For example, the ''N'' in `ypcat`(1N) indicates that this manual page describes a networking command. Here is an explanation of each subcategory:

  1C Communications commands, such as `cu` and `tip`.

  1G Graphics commands, such as `graph` and `tplot`.

  1N Networking commands, such as those that help support various networking subsystems, including the Network File System (NFS), Remote Process Control (RPC) subsystem, and Internet subsystem.

- Section 6—Games
  This section contains all of the games provided with A/UX, such as `cribbage` and `worms`.

A/UX Programmer's Reference contains Sections 2 through 5 and Section 7.

- Section 2—System Calls
  This section describes the services provided by the A/UX system kernel, including the C language interface. It includes two special categories. Where applicable, these categories are indicated by the letter designation that follows the section number. For example, the ''N'' in `connect`(2N) indicates that this manual page describes a networking command. Here is an explanation of each subcategory:

2N	Networking system calls
2P	POSIX system calls

- Section 3—Subroutines
  This section describes the available subroutines. The binary versions of these subroutines are in the system libraries in the `/lib` and `/usr/lib` directories. The section includes seven special categories. Where applicable, these categories are indicated by the letter designation that follows the section number. For example, the ''N'' in `mount`(3N) indicates that this manual page describes a networking command. Here is an explanation of each subcategory:

3C	C and assembly-language library routines
3F	Fortran library routines
3M	Mathematical library routines
3N	Networking routines
2P	POSIX routines
3S	Standard I/O library routines
3X	Miscellaneous routines

- Section 4—File Formats
  This section describes the structure of some files, but does not include files that are used by only one command (such as the assembler's intermediate files). The C language `struct` declarations corresponding to these formats are in the `/usr/include` and `/usr/include/sys` directories. There is one special category in this section, indicated by the letter designation ''N'' following the section number:

4N Networking formats

- Section 5—Miscellaneous Facilities
  This section describes various character sets, macro packages, and other miscellaneous facilities. There are two special categories in this section. Where applicable, these categories are indicated by the letter designation that follows the section number. For example, the "P" in `tcp`(1P) indicates a protocol. Here is an explanation of each subcategory:

  5F Protocol families

  5P Protocol descriptions

- Section 7—Drivers and Interfaces for Devices
  This section describes the drivers and interfaces through which devices are normally accessed. Access to one or more disk devices is fairly transparent when you are working with them in terms of files. When you want to use A/UX commands to communicate with devices more directly, at a level beyond the moderation of file systems, device files serve your needs. Such a level of communication permits you to request more explicit operating modes that may be supported by a disk (such as accessing disk partition maps), or that may be supported by other types of devices, such as tape drives and modems. For example, you can access a tape device in automatic-rewind mode as described in `tc`(7).

A/UX System Administrator's Reference contains Sections 1M and 8.

- Section 1M—System Maintenance Commands
  This section describes system maintenance programs such as `fsck` and `mkfs`.

- Section 8—A/UX Startup Shell Commands
  This section describes the commands that are available from within the A/UX Startup shell. This section includes detailed descriptions of the commands that contribute to the boot process and those that help with the maintenance of inactive file systems.

For more information

To find out where you need to go for more information about how to use A/UX, see *Road Map to A/UX*. This guide contains descriptions of each A/UX guide and ordering information for all the guides in the A/UX documentation suite.

the conventions used in all the A/UX reference books.

Keys and key combinations

Certain keys on the keyboard have special names. These modifier and character keys, often used in combination with other keys, perform various functions. In this book, the names of these keys appear in the format of an initial capital letter followed by small capital letters.

Here is a list of the most common key names:

CAPS LOCK	ENTER	SHIFT
COMMAND	ESCAPE	SPACE BAR
CONTROL	OPTION	TAB
DELETE	RETURN	

Sometimes two or more key names are joined by hyphens. The hyphens indicate that you press these keys simultaneously to perform a specific function. For example,

Press CONTROL-K

means ''While holding down the CONTROL key, press the K key.''

Terminology

In A/UX manuals, a certain term can represent a specific set of actions. For example, the word ''enter'' indicates that you type a series of characters, then press the RETURN key. The instruction

Enter `whoami`.

means ''Type `whoami`, then press the RETURN key.'' (If you entered this text at a command prompt, the system would respond by displaying your current account name.)

Here is a list of common terms and their corresponding actions.

Here's an example:

Type `date` on the command line and press RETURN.

This instruction means that you should type the word ''date'' exactly as shown, then press the RETURN key.

After you press RETURN, text such as this will appear on the screen:

```
Fri Nov  1 11:15:43 PST 1991
```

In this case, the Courier font is used to represent exactly what appears on the screen.

All A/UX manual page names are shown in the Courier font. For example, `ls`(1) indicates that `ls` is the name of a manual page that occurs in Section 1. More information about the use of the Courier font in manual pages is given in ''Styling of A/UX Command Elements'' and in ''Styling of Cross-References to Manual Pages'' later in this preface.

Font styles

Italics are used to indicate that a word or set of words is a placeholder for part of a command line. Here is a sample command syntax illustration:

`cat` *file*

The italicized term *file* is a placeholder for the name of a file. If you wanted to display the contents of a file named `Elvis`, you would type the filename `Elvis` in place of *file*. In other words, you would enter

```
cat Elvis
```

Styling of A/UX command elements

A/UX commands are entered in accordance with their command syntax. A typical A/UX command line includes the command name first, followed by options and arguments. For example, here is an illustration of the syntax for the `wc` command:

`wc [-l] [-w]` *file...*

In this syntax illustration, `wc` is the command, `-l` and `-w` are options, and *file* is an argument.

A ''command option'' modifies the action of a command, often by changing its mode of operation (such as read mode or write mode).

Styling of cross-references to manual pages

The manual pages are organized primarily in terms of sections, and secondarily in terms of books for different audiences. The standard A/UX cross-reference notation leaves out the book title, but refers to the section designation:

item(*section*)

where *item* is the name of the command, subroutine, or other A/UX provision, and *section* is the section where the manual page resides.

For example,

`cat`(1)

refers to the command `cat`, which is described in Section 1, which is in *A/UX Command Reference.*

As a guide to the location of sections, you can refer to the general table of contents of each of the primary reference manuals, or to ''Current Organization of Sections into Books'' later in this preface. (The binder spines are also labeled with the section numbers, and occasionally section subdivisions, that are in each binder.)

Note also that there are a number of subcategory designations that can follow the digit reference in (1), (2), (3), (4), and (5), such as (1N). Detailed explanations of these subcategory designations are provided later in this preface.

Previous organization of sections into books

You may be curious about the logic behind the numbering of sections. The derivation of this numbering is much clearer when you realize that originally there was only one reference manual, the *UNIX User Manual.* In fact the manual pages were once considered the primary UNIX documentation, and the other books were originally considered supplements.

In the early days, all the manual pages easily fit into one book, in sections numbered 1 through 8. Section 8 originally contained the manual pages that are now located in Section 1M.

With the expansion of the original sections as UNIX grew, it became necessary to split the original book into several books, and this was done according to the audience they served. However, the original section numbering was preserved after the split because by then each number had come to have a particular meaning to UNIX users.

Table of Contents

Section 1: User Commands (Q-Z)

NAME

`query` — queries the user for input

SYNOPSIS

`query` [`-t`[*seconds*]] [`-r`[*response*]] [`-m`]

ARGUMENTS

`-m` Watches for a mouse click. If the mouse does get clicked, exit status 2 is returned. Note: This option will be ignored if any other program (such as a toolbox application) is currently using the mouse.

`-r`[*response*]
: Changes the default response to the given response. The default response is `y` if not set with this option. This option is only useful in conjunction with the `-t` option.

`-t`[*seconds*]
: Times out after the given seconds. If no input has been seen by this time, `query` will echo the default response value to standard output and standard error.

DESCRIPTION

By default, `query` reads a line from standard input and echoes it to standard output.

STATUS MESSAGES AND VALUES

Exit status is 0 if everything is OK, 1 for usage error, 2 if mouse is pressed when `query -m` is in use.

FILES

`/etc/query`
: Executable file

SEE ALSO

`line`(1)

NAME

`rcp` — copies files between two systems

SYNOPSIS

`rcp` *file1 file2*

`rcp [-r]` *file... directory*

ARGUMENTS

directory
: Specifies the directory into which the files will be copied.

file Specifies the file that is to be copied into the given directory.

file1
: Specifies the file in the current directory that is to be copied into a remote directory.

file2
: Specifies the new file that was copied in the remote directory.

`-r` Causes `rcp` to copy each subtree rooted at that name if any of the source files are directories; in this case, the destination host is used.

DESCRIPTION

`rcp` copies files between machines. Each *file* or *directory* argument is either a remote filename of the form *rhost*`:`*path* or a local filename (containing no `:` characters, or a `/` before a `:`).

By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the `umask`(2) on the destination host is used.

If *path* is not a full pathname, it is interpreted relative to the login directory on *rhost*. A *path* on a remote host may be quoted (using \, ", or ') so that the metacharacters are interpreted remotely.

`rcp` does not prompt for passwords; the current local user name must exist on *rhost* and allow remote command execution via `remsh`(1N).

`rcp` handles third party copies, where neither source nor target files are on the current machine. Host names may also take the form *rname*`@`*rhost* to use *rname* rather than the current user name on the remote host.

EXAMPLES

The command

```
rcp recipe doc:cake
```

copies the file `recipe` from the current directory and renames it as `cake` in the remote login directory on `doc`.

The command

```
rcp -r doc:Test .
```

creates a new directory `Test` below the current (local) directory. The local `Test` contains copies of every file and subdirectory contained in the remote `Test` on the machine `doc`. Note that both examples assume that there is a login directory on `doc` and that permissions are set correctly. See the network issues appendix in *A/UX Networking Essentials* for more information.

LIMITATIONS

`rcp` doesn't detect all cases where the target of a copy might be a file when only a directory should be legal.

`rcp` is confused by any output generated by commands in a `.login`, `.profile`, or `.cshrc` file on the remote host.

FILES

`/usr/bin/rcp`
  Executable file

SEE ALSO

`cp`(1), `ftp`(1N), `remsh`(1N), `rlogin`(1N)

A/UX Networking Essentials

NAME

`rcs` — creates new RCS files or changes attributes of existing RCS files

SYNOPSIS

`rcs` [`-a`*logins*]] [`-A`*oldfile*] [`-c`*string*] [`-e`[*logins*]] [`-i`] [`-l`[*rev*]] [`-L`]
[`-n`*name*[`:`*rev*]] [`-N`*name*[`:`*rev*]] [`-o`*range*] [`-q`] [`-s`*state*[`:`*rev*]]
[`-t`[*txtfile*]] [`-u`[*rev*]] [`-U`] *files*

ARGUMENTS

`-a`*logins*
: Appends the login names appearing in the comma-separated list *logins* to the access list of the RCS file.

`-A`*oldfile*
: Appends the access list of *oldfile* to the access list of the RCS file.

`-c`*string*
: Sets the comment leader to *string*. The comment leader is printed before every log-message line generated by the keyword `$Log$` during checkout (see `co`). This is useful for programming languages without multiline comments. During `rcs -i` or initial `ci`, the comment leader is guessed from the suffix of the working file.

`-e`[*logins*]
: Erases the login names appearing in the comma-separated list *logins* from the access list of the RCS file. If *logins* is omitted, the entire access list is erased.

files Specifies the RCS files to be affected.

`-i` Creates and initializes a new RCS file, but does not deposit any revision. If the RCS file has no path prefix, `rcs` tries to place it, first into the subdirectory `./RCS` and then into the current directory. If the RCS file already exists, an error message is printed.

`-l`[*rev*]
: Locks the revision with number *rev*. If a branch is given, the latest revision on that branch is locked. If *rev* is omitted, the latest revision on the trunk is locked. Locking prevents overlapping changes. A lock is removed with `ci` or `rcs -u`.

`-L` Sets locking to `strict`. Strict locking means that the owner of an RCS file is not exempt from locking for checkin. This option should be used for files that are shared.

`-n`*name*[`:`*rev*]
: Associates the symbolic name *name* with the branch or revision *rev*. `rcs` prints an error message if *name* is already associated with another number. If *rev* is omitted, the symbolic name is deleted.

`-N`*name*[`:`*rev*]
: Overrides a previous assignment of *name*. Otherwise, this option is the same as the `-n` option.

`-o`*range*
: Deletes (outdates) the revisions given by *range*. A range consisting of a single revision number means that revision. A range consisting of a branch number means the latest revision on that branch. A range of the form *rev1*`-`*rev2* means revisions *rev1* to *rev2* on the same branch, `-`*rev* means from the beginning of the branch containing *rev* up to and including *rev*, and *rev*`-` means from revision *rev* to the end of the branch containing *rev*. None of the outdated revisions may have branches or locks.

`-q` Specifies quiet mode; diagnostics are not printed.

`-s`*state*[`:`*rev*]
: Sets the state attribute of the revision *rev* to *state*. If *rev* is omitted, the latest revision on the trunk is assumed. If *rev* is a branch number, the latest revision on that branch is assumed. Any identifier is acceptable for *state*. A useful set of states is `Exp` (for experimental), `Stab` (for stable), and `Rel` (for released). By default, `ci` sets the state of a revision to `Exp`.

`-t`[*txtfile*]
: Writes descriptive text into the RCS file and deletes the existing text. If *txtfile* is omitted, `rcs` prompts the user for text supplied from the standard input, terminated with a line containing a single `.` or CONTROL-D. Otherwise, the descriptive text is copied from the file *txtfile*. If the `-i` option is present, descriptive text is requested even if `-t` is not given. The prompt is suppressed if the standard input is not a terminal.

`-u`[*rev*]
: Unlocks the revision with number *rev*. If a branch is given, the latest revision on that branch is unlocked. If *rev* is omitted, the latest lock held by the caller is removed. Normally, only the locker of a revision may unlock it. Anyone else unlocking a revision breaks the lock. This causes a mail message to be sent to the original locker. The message contains a commentary solicited from the breaker. The commentary is terminated with a line containing a single `.` or CONTROL-D.

`-U` Sets locking to non-strict. Non-strict locking means that the owner of a file need not lock a revision for checkin. This option should *not* be used for files that are shared. The default (`-L` or `-U`) is determined by your system administrator.

DESCRIPTION

`rcs` creates new RCS files or changes attributes of existing ones. An RCS file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For `rcs` to work, the caller's login name must be on the access list, unless the access list is empty, the caller is the owner of the file or the superuser, or the `-i` option is present.

Files ending in `,v` are RCS files, and all others are working files. If a working file is given, `rcs` tries to find the corresponding RCS file, first in directory `./RCS` and then in the current directory, as explained in `co(1)`.

The caller of the command must have read/write permission for the directory containing the RCS file and read permission for the RCS file itself. `rcs` creates a semaphore file in the same directory as the RCS file to prevent simultaneous update. For changes, `rcs` always creates a new file. On successful completion, `rcs` deletes the old one and renames the new one. This strategy makes links to RCS files useless.

STATUS MESSAGES AND VALUES

The RCS filename and the revisions outdated are written to the diagnostic output. The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 if otherwise.

NOTES

Author: Walter F. Tichy, Purdue University, West Lafayette, IN 47907. Copyright © 1982 by Walter F. Tichy.

SEE ALSO

`ci`(1), `co`(1), `ident`(1), `rcsdiff`(1), `rcsintro`(1), `rcsmerge`(1), `rlog`(1)

`sccstorcs`(1M) in *A/UX System Administrator's Reference*

`rcsfile`(4) in *A/UX Programmer's Reference*

Walter F. Tichy, ''Design, Implementation, and Evaluation of a Revision Control System,'' in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, September 1982

NAME

`rcsdiff` — compares RCS revisions

SYNOPSIS

`rcsdiff [-b] [-c] [-e] [-f] [-h] [-i] [-n] [-t] [-w] [-r`*rev1*`]`
`[-r`*rev2*`]` *file ...*

ARGUMENTS

`-b` Causes trailing blanks (spaces and tabs) to be ignored, and other strings of blanks to compare equal.

`-c` Produces a `diff` with lines of context. The default is to present 3 lines of context and may be changed, e.g., to 10, by `-c10`. With the `-c` option, the output format is modified slightly: the output beginning with identification of the files involved and their creation dates and then each change is separated by a line with a dozen `*`'s. The lines removed from *file1* are marked with ''–''; those added to *file2* are marked ''+''. Lines which are changed from one file to the other are marked in both files with ''!''.

`-e` Produces a script of *a*, *c*, and *d* commands for the editor `ed`, which will recreate *file2* from *file1*. In connection with the `-e`, option, 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
```

Extra commands are added to the output when comparing directories with the `-e,` option so that the result is a `sh`(1) script for converting text files which are common to the two directories from their state in *dir1* to their state in *dir2*. Since such a shell script is useful only in a file that you may run on other files, it is best to redirect the output of this command into a file.

`-f` Produces a script similar to that of the `-e,` option not useful with `ed,` and in the opposite order.

file Specifies the RCS file which has different versions that are to be compared.

`-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.

`-i` Ignores lower and upper case character differences.

`-n` Produces a script similar to that of the `-f` option but with a count of changed lines added.

`-t` Expands tabs in output lines, preserving correct indentation of the source text.

`-w` Causes spaces and tabs to be ignored.

`-r`*rev1*
Specifies the first revision of the RCS file that is compared with *rev2.* If this argument is given, but not *rev2*, `rcsdiff` compares *rev1* of the RCS file with the contents of the corresponding working file.

`-r`*rev2*
Specifies the second revision of the RCS file that is compared with *rev1.*

DESCRIPTION

`rcsdiff` runs `ucbdiff(1)` to compare two revisions of each RCS file given. A filename ending in `,v` is an RCS filename, otherwise a working filename. `rcsdiff` derives the working filename from the RCS filename and vice versa, as explained in `co`(1). Pairs consisting of both an RCS and a working filename may also be specified.

If both *rev1* and *rev2* are omitted, `rcsdiff` compares the latest revision on the trunk with the contents of the corresponding working file. This is useful for determining what was changed since the last checkin.

If both *rev1* and *rev2* are given, `rcsdiff` compares revisions *rev1* and *rev2* of the RCS file.

Both *rev1* and *rev2* may be given numerically or symbolically.

EXAMPLES

The command:

```
rcsdiff f.c
```

runs `ucbdiff` on the latest trunk revision of RCS file `f.c,v` and the contents of working file `f.c`.

NOTES

Author: Walter F. Tichy, Purdue University, West Lafayette, IN 47907. Copyright © 1982 by Walter F. Tichy.

SEE ALSO

`ci`(1), `co`(1), `ucbdiff`(1), `ident`(1), `rcs`(1), `rcsintro`(1), `rcsmerge`(1), `rlog`(1)

`rcsfile`(4) in *A/UX Programmer's Reference*

Walter F. Tichy, ''Design, Implementation, and Evaluation of a Revision Control System,'' in *Proceedings of the 6th International Conference on Software Engineering,* IEEE, Tokyo, Sept. 1982

NAME

`rcsmerge` — merges two versions of an RCS file

SYNOPSIS

`rcsmerge -r`*rev1* [`-r`*rev2*] [`-p`] *file*

ARGUMENTS

file Specifies the RCS file whose versions are to be merged.

`-p` Prints the results of the merge on the standard output; otherwise the results overwrite the working file.

`-r`*rev1*
Specifies the first revision of the file to be merged. This argument cannot be omitted.

`-r`*rev2*
Specifies the next revision of the file to be merged. If this argument is omitted, the latest revision on the trunk is assumed.

DESCRIPTION

`rcsmerge` incorporates the changes between *rev1* and *rev2* of an RCS file into the corresponding working file.

A filename ending in '`,v`' is an RCS filename, otherwise a working filename. `rcsmerge` derives the working filename from the RCS filename and vice versa, as explained in `co(1)`. A pair consisting of both an RCS and a working filename may also be specified.

Both *rev1* and *rev2* may be given numerically or symbolically.

The `rcsmerge` program prints a warning if there are overlaps and delimits the overlapping regions as explained in `co -j`. The command is useful for incorporating changes into a checked-out revision.

EXAMPLES

Suppose you have released revision 2.8 of `f.c`. Assume further that you just completed revision 3.4, when you receive updates to release 2.8 from someone else. To combine the updates to 2.8 and your changes between 2.8 and 3.4, put the updates to 2.8 into file `f.c` and execute the command:

```
rcsmerge -p -r2.8 -r3.4 f.c > f.merged.c
```

Then examine `f.merged.c`. Alternatively, if you want to save the updates to 2.8 in the RCS file, check them in as revision 2.8.1.1 and execute `co -j`:

```
ci -r2.8.1.1 f.c
co -r3.4 -j2.8:2.8.1.1 f.c
```

As another example, the following command undoes the changes between revision 2.4 and 2.8 in your currently checked out revision in `f.c`.

```
rcsmerge -r2.8 -r2.4 f.c
```

Note the order of the arguments and that `f.c` will be overwritten.

LIMITATIONS

`rcsmerge` does not work for files that contain lines with a single '`.`'.

NOTES

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.0; Release Date: 83/01/15.
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

`ci`(1), `co`(1), `merge`(1), `ident`(1), `rcs`(1), `rcsdiff`(1), `rlog`(1)

`rcsfile`(4) in *A/UX Programmer's Reference*

Walter F. Tichy, ''Design, Implementation, and Evaluation of a Revision Control System,'' in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982

NAME

`rcvhex` — receives and converts Motorola S-records from a port to a file

SYNOPSIS

`rcvhex` [`-p` *port*] [`-c` *command*] *file*

ARGUMENTS

`-c` *command*
: Ships the specified command (in quotes) over the remote port; the default is to not ship anything.

file Specifies the file to be created by the `rcvhex` command.

`-p` *port*
: Specifies an alternate port for reception; the default port is `/dev/tty0`.

DESCRIPTION

`rcvhex` translates Motorola S-records shipped from a port into a file.

The file's starting address must be zero and successive records must be sequential.

FILES

`/usr/bin/rcvhex`
: Executable file

SEE ALSO

`hex(1)`

NAME

`rdist` — distributes remote files

SYNOPSIS

`rdist` [`-b`] [`-d`*var=value*] [`-f`*distfile*] [`-h`] [`-i`] [`-m`*host*] [`-n`] [`-q`] [`-R`] [`-v`] [`-w`] [`-y`] [*name*]...

`rdist` [`-b`] `-c` *name...* [`-h`] [`-i`] [`-n`] [`-q`] [`-R`] [`-v`] [`-w`] [`-y`] [*login*`@`] *host* [`:`*dest*]

ARGUMENTS

`-b` Specifies binary comparison. Perform a binary comparison and update files if they differ rather than comparing dates and sizes.

`-c` *name*

Forces `rdist` to interpret the remaining arguments as a small *distfile*. The equivalent *distfile* is as follows.

```
( name...) ->[login@]host
install [dest];
```

`-d`*var=value*

Defines *var* to have *value*. The `-d` option is used to define or override variable definitions in the *distfile*. *value* can be the empty string, one name, or a list of names surrounded by parentheses and separated by tabs and/or spaces.

`:`*dest*

Specifies the destination system on which the files will be distributed.

`-f`*distfile*

Causes the program to first look for *distfile*, then *Distfile* to use as the input, if this option is not present. If no names are specified on the command line, `rdist` will update all of the files and directories listed in *distfile*. Otherwise, the argument is taken to be the name of a file to be updated or the label of a command to execute. If label and filenames conflict, it is assumed to be a label. These may be used together to update specific files using specific commands.

`-h` Follows symbolic links. Copy the file that the link points to rather than the link itself.

host Specifies the local host system.

`-i` Ignores unresolved links. `rdist` will normally try to maintain the link structure of files being transferred and warn the user if all the links cannot be found.

login `@`

Specifies the login name used on the destination host. The login name is the same as the local host unless the destination name is of the format *login@host*.

-m*host*
: Limits which machines are to be updated. Multiple -m arguments can be given to limit updates to a subset of the hosts listed in *distfile*.

-n Prints the commands without executing them. This flag option is useful for debugging *distfile*.

-q Specifies quiet mode. Files that are being modified are normally printed on standard output. The -q option suppresses this.

-R Removes extraneous files. If a directory is being updated, any files that exist on the remote host that do not exist in the master directory are removed. This is useful for maintaining truly identical copies of directories.

-v Verifies that the files are up to date on all the hosts. Any files that are out of date will be displayed but no files will be changed nor any mail sent.

-w Specifies whole mode. The whole filename is appended to the destination directory name. Normally, only the last component of a name is used when renaming files. This will preserve the directory structure of the files being copied instead of flattening the directory structure. For example, renaming a list of files such as (dir1/f1 dir2/f2) to dir3 would create files dir3/dir1/f1 and dir3/dir2/f2 instead of dir3/f1 and dir3/f2.

-y Specifies younger mode. Files are normally updated if their *mtime* and *size* (see stat(2)) disagree. The -y option causes rdist not to update files that are younger than the master copy. This can be used to prevent newer copies on other hosts from being replaced. A warning message is printed for files which are newer than the master copy.

DESCRIPTION

rdist is a program to maintain identical copies of files over multiple hosts. It preserves the owner, group, mode, and *mtime* (see stat(2)) of files if possible and can update programs that are executing. rdist reads commands from *distfile* to direct the updating of files and/or directories. If *distfile* is ''-'', the standard input is used.

Replace *distfile* with a sequence of entries that specify the files to be copied, the destination hosts, and what operations to perform to do the updating. Each entry has one of the following formats.

<variable name> = *<name*list*>*
[*label*:] *<source list>* -> *<destination list>* *<command list>*
[*label*:] *<source list>* :: *<time_stamp file>* *<command list>*

The first format is used for defining variables. The second format is used for distributing files to other hosts. The third format is used for making lists of files that have been changed since some given date. The ''source list'' specifies a list of files and/or directories on the local host which are to be used as the master copy for distribution. The ''destination list'' is the list of hosts to which these files are to be copied. Each file in the source list is added to a list of changes if the file is out of date on the host which is being updated (second format) or the file is newer than the time stamp file (third format).

Labels are optional; they are used to identify a command for partial updates.

Newlines, tabs, and blanks are only used as separators and are otherwise ignored. Comments begin with ''#'' and end with a newline.

Variables to be expanded begin with ''$'' followed by one character or a name enclosed in braces (see the examples at the end).

The source and destination lists have the following format:

<name>
or
'' ('' *<zero or more names separated by white-space>* '') ''

The shell metacharacters ''['', '']'', ''{'', ''}'', ''*'', and ''?'' are recognized and expanded (on the local host only) in the same way as `csh`(1). They can be escaped with a backslash. The ~ character is also expanded in the same way as `csh` but is expanded separately on the local and destination hosts. When the `-w` option is used with a filename that begins with ~, everything except the home directory is appended to the destination name. File names which do not begin with // or ~ use the destination user's home directory as the root directory for the rest of the filename.

The command list consists of zero or more commands of the following format.

`install`
    <options>opt_dest_name;
`notify`
    <name list> ;
`except`
    <name list> ;
`except_pat`
    <pattern list> ;
`special`
    <name list>string;

The `install` command is used to copy out of date files and/or directories. Each source file is copied to each host in the destination list. Directories are recursively copied in the same way. The parameter, ''opt_dest_name'', is optional and is used to rename files. If no `install` command appears in the command list or the destination name is not specified, the source filename is used. Directories in the pathname will be created if they do not exist on the remote host. To help prevent disasters, a nonempty directory on a target host will never be replaced with a regular file or a symbolic link. However, under the `-R` option, a nonempty directory will be removed if the corresponding filename is completely absent on the master host. The *options* are `-R`, `-h`, `-i`, `-v`, `-w`, `-y`, and `-b` and have the same semantics as options on the command line except they only apply to the files in the source list.

The `notify` command is used to mail the list of files updated (and any errors that may have occurred) to the listed names. If no `@` appears in the name, the destination host is appended to the name (e.g., *name1*`@`*host*, *name2*`@`*host*, . . .) .

The `except` command is used to update all of the files in the source list `except` for the files listed in ''name list''. This is usually used to copy everything in a directory except certain files.

The `except_pat` command is like the `except` command except that ''pattern list'' is a list of regular expressions (see `ed`(1) for details). If one of the patterns matches some string within a filename, that file will be ignored. Note that since `\` is a quote character, it must be doubled to become part of the regular expression. Variables are expanded in ''pattern list'' but not shell file pattern matching characters. To include a `$`, it must be escaped with `\`.

The `special` command is used to specify `sh`(1) commands that are to be executed on the remote host after the file in ''name list'' is updated or installed. If the ''name list'' is omitted then the shell commands will be executed for every file updated or installed. The shell variable `FILE` is set to the current filename before executing the commands in ''string.'' A string starts and ends with quotes ( ) and can cross multiple lines in *distfile.* Multiple commands to the shell should be separated by `;`. Commands are executed in the user's home directory on the host being updated. The `special` command can be used to rebuild private databases, etc. after a program has been updated.

EXAMPLES

The following is a small example.

```
HOSTS = ( matisse root@arpa)

FILES = ( /bin /lib /usr/bin /usr/games
          /usr/include/{*.h,{stand,sys,vax*,pascal,machine}/*.h}
          /usr/lib /usr/man/man? /usr/ucb /usr/local/rdist )

    EXLIB = ( Mail.rc aliases aliases.dir aliases.pag crontab dshrc
          sendmail.cf sendmail.fc sendmail.hf sendmail.st uucp vfont )

    ${FILES} -> ${HOSTS}
          install ;
          except /usr/lib/${EXLIB} ;
          except /usr/games/lib ;
          special /usr/lib/sendmail "/usr/lib/sendmail -bz" ;

    srcs:
    /usr/src/bin -> arpa
       except_pat ( \\.o\$ /SCCS\$ ) ;

    IMAGEN = (ips dviimp catdvi)

    imagen:
    /usr/local/${IMAGEN} -> arpa
       install /usr/local/lib ;
       notify ralph ;

    ${FILES} :: stamp.cory
       notify root@cory ;
```

STATUS MESSAGES AND VALUES

A complaint about mismatch of rdist version numbers may really stem from some problem with starting your shell, e.g., you are in too many groups.

LIMITATIONS

Source files must reside on the local host where `rdist` is executed.

There is no easy way to have a special command executed after all files in a directory have been updated.

Variable expansion only works for name lists; there should be a general macro facility.

`rdist` aborts on files which have a negative *mtime* (before Jan 1, 1970).

There should be a ''force'' option to allow replacement of nonempty directories by regular files or `symlinks`. A means of updating file modes and owners of otherwise identical files is also needed.

FILES

`/usr/bin/rdist`
: Executable file

`/tmp/rdist*`
: Temporary file for update lists

SEE ALSO

sh(1), csh(1)

stat(2) in *A/UX Programmer's Reference*

See ed(1)

NAME

`refer` — finds and inserts literature references in documents

SYNOPSIS

`refer` [`-a`[*n*]] [`-b`] [`-B`[*l*`.`*m*]] [`-c` *keys*] [`-e`] [`-f`*n*] [`-k`*x*] [`-l`[*m*`,`*n*]] [`-n`] [`-p` *bib*] [`-P`] [`-s` *keys*] [`-S`] [*file*]...

ARGUMENTS

`-a`[*n*]
: Reverses the first *n* author names, for example, Jones, J. A. instead of J. A. Jones. If *n* is omitted, all author names are reversed.

`-b` Specifies bare mode: do not put any flags in text (neither numbers nor labels).

`-B`[*l*`.`*m*]
: Specifies bibliography mode. Takes a file composed of records separated by blank lines and turns them into `troff` input. Label *l* is turned into the macro `.`*m* with *l* defaulting to `%X` and `.`*m* defaulting to `.AP` (annotation paragraph).

`-c` *keys*
: Capitalizes (with CAPS SMALL CAPS) the fields whose key-letters are in *keys*.

`-e` Accumulates the references until a sequence of the form

```
.[
$LIST$
.]
```

is encountered, and then writes out all references collected so far and collapses the references to the same source. Usually, the references are left where encountered.

`-f`*n*
: Sets the initial footnote number to *n* instead of the default of 1. With labels rather than numbers, this is a no-op.

file Specifies the file to be searched.

`-k`*x* Uses labels as specified in a reference data line beginning `%`*x*`,` instead of numbering references. By default, *x* is `L`.

`-l`[*m*`,`*n*]
: Uses labels made from the senior author's last name and the year of publication, instead of numbering references. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either *m* or *n* is omitted, the entire name or date is used.

`-n` Does not search `/usr/dict/papers/Ind`, the default file. If there is a `REFER` environment variable, the specified file is searched instead of the default file; in this case the `-n` has no effect.

`-p` *bib*
: Takes the next argument *bib* as a file of references to be searched. The default file is searched last.

`-P` Places punctuation marks (. , : ; ? !) after the reference signal rather than before. (Periods and commas used to be done with strings.)

`-s` *keys*
: Sorts references by fields whose keyletters are in the *keys* string; permutes the reference numbers in text accordingly. This option implies `-e`. The keyletters in *keys* may be followed by a number to indicate how many such fields are used, with + taken as a very large number. The default is `AD`, which sorts on the senior author and then date. To sort, for example, on all authors and then, the title, use `-sA+T`.

`-S` Produces references in the Natural or Social Science format.

DESCRIPTION

`refer` is a preprocessor for `nroff`(1) or `troff`(1) that finds and formats references for footnotes or endnotes. It is also the base for a series of programs designed to index, search, sort, and print stand-alone bibliographies, or other data entered in the appropriate form.

Given an incomplete citation with sufficiently precise keywords, `refer` searches a bibliographic database for references containing these keywords anywhere in the title, author, journal, and so forth. The input file (or standard input) is copied to standard output, except for lines between `.[` and `.]` delimiters, which are assumed to contain keywords and are replaced by information from the bibliographic database. The user may also search different databases, override particular fields, or add new fields. The reference data, from whatever source, is assigned to a set of `troff` strings. Macro packages such as `ms`(5) print the finished reference text from these strings. By default, references are flagged by footnote numbers.

To use your own references, put them in the format described below. When `refer` is used with the `eqn`, `neqn`, or `tbl` preprocessors, `refer` should be first, to minimize the volume of data passed through pipes.

The `refer` preprocessor and associated programs expect input from a file of references composed of records separated by blank lines. A record is a set of lines (fields), each containing one kind of information. Fields start on a line beginning with a `%`, followed by a keyletter, then a blank, and finally the contents of the field, which continue until the next line starting with `%`. The output ordering and formatting of fields is controlled by the macros specified for `nroff`/`troff` (for footnotes and endnotes) or roffbib (for stand-alone bibliographies). For a list of the most common key-letters and their corresponding fields, see addbib(1).

EXAMPLES

An example of a refer entry is given below:

```
%A      T. Monroe
%T      Creating Inverted Indexes
%B      Text Processing Guide
%V      2.6b
%I      Data Systems
%C      Berkeley, California
%D      1998
```

LIMITATIONS

Blank spaces at the end of lines in bibliography fields will cause the records to sort and reverse incorrectly. Sorting large numbers of references causes a core dump.

FILES

/usr/ucb/refer
: Executable file

/usr/dict/papers
: Directory containing default publication lists

/usr/lib/refer
: Directory containing companion programs

SEE ALSO

addbib(1), indxbib(1), lookbib(1), roffbib(1), sortbib(1)

NAME

regcmp — compiles regular expressions with a file

SYNOPSIS

regcmp [-] *file...*

ARGUMENTS

-	Places the output in a file suffixed .c.
file	Specifies the file to be compiled.

DESCRIPTION

regcmp, in most cases, precludes the need for calling regcmp from C programs. This saves on both execution time and program size. The command regcmp compiles the regular expressions in *file* and places the output in *file*.i.

The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of regcmp is C source code. Compiled regular expressions are represented as extern char vectors. *file*.i files may thus be included into C programs, or *file*.c files may be compiled and later loaded.

In the C program which uses the regcmp output, *regex*(*abc*,*line*) will apply the regular expression named *abc* to *line*. The status messages for this command are self-explanatory.

EXAMPLES

This C program:

```
name“([A-Za-z][A-Za-z0-9_]*)$0”

telno
“,1}([2-9][01][1-9])$0){0,1} *”
“([2-9][0-9]{2})$1[ -]{0,1}”
“([0-9]{4})$2”
```

that uses the regcmp output

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named telno to line.

FILES

/usr/bin/regcmp
    Executable file

SEE ALSO

cc(1), lex(1)

regcmp(3X) in *A/UX Programmer's Reference*

NAME

`remsh` — invokes to a shell on a remote system

SYNOPSIS

`remsh` *rhost* [`-l` *username*] [`-n`] [*command*]

ARGUMENTS

command
: Specifies the remote command to be executed. If you omit *command*, then instead of executing a single command, you are logged in on the remote host using `rlogin`(1N).

`-l` *username*
: Specifies a different remote user name. By default, the remote user name that is used is the same as your local user name. The remote account must have its `rhosts` file set up to grant you permission to log in without prompting you for the password; no provision is made for specifying a password with a command.

`-n` Redirects the standard input to `/dev/null`.

rhost
: Specifies the remote host system to connect to.

DESCRIPTION

`remsh` connects to a specified remote host, *rhost,* and executes a specified remote command (*command*) via a local network. On the remote side, you get whatever shell is set up for that account. The `remsh` program copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit, and terminate signals are propagated to the remote command; `remsh` normally terminates when the remote command does.

Shell metacharacters that are not quoted are interpreted on the local machine while quoted metacharacters are interpreted on the remote machine. Thus the command

```
remsh rhost cat remotefile >> localfile
```

appends the remote file `remotefile` to the local file `localfile`, while

```
remsh rhost cat remotefile ">>" remotefile.2
```

appends `remotefile` to `remotefile.2`.

Host names are given in the file `/etc/hosts`. Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames. The remote host system names for local machines may also be commands in the directory

/usr/hosts; these names must be linked to the remsh binaries. If you put this directory in your search path, then the remsh may be omitted, as in the second form of the command, above.

Using remsh, you cannot run an interactive command (like vi(1)); instead, use rlogin(1N).

FILES

/usr/bin/remsh
  Executable file
/etc/hosts
  Host file
/usr/hosts/*
  Directory containing host files

SEE ALSO

rlogin(1N), telnet(1C)

A/UX Networking Essentials

See `tset`(1)

NAME

`rev` — reverses characters within each line of text

SYNOPSIS

`rev` [*file*]...

ARGUMENTS

file Specifies the file that will be copied.

DESCRIPTION

`rev` copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

FILES

`/usr/ucb/rev`
Executable file

NAME

`rez` — compiles Macintosh resource files from source code

SYNOPSIS

`rez` [`-a`] [`-align` *word-type*] [`-c` *reator*] [`-d` *macro-assignment*]...
[`-i` *include-dir*]... [`-o` *output-file*] [`-ov`] [`-p`] [`-rd`] [`-ro`]
[`-s` *res-include-dir*]... [`-t` *type*] [`-u` *macro*]
[*resource-description-file*]...

ARGUMENTS

`-align word`
`-align longword`
Align resources along word or longword boundaries. This alignment may allow the Resource Manager to load these resources more quickly. The `-align` option is ignored when the `-a` option is in effect.

`-a` Appends the output from `rez` to the output file instead of replacing the output file.

`-c0`*creator*
Specifies the creator attribute for the compiled resource file.

`-d` *macro-assignment*
Declares a macro and its value in the form

macro=value

This assignment is equivalent to the following form of preprocessor request:

`#define` *macro* [*value*]

If *value* is omitted, then *macro* is set to the null string. (The macro is still considered to be defined.)

`-i` *include-dir*
Searches the specified directory for include files. The system searches directories according to the order of appearance of any number of `-i` options in the command line. To reach the include files provided with the A/UX Toolbox, use this command: (Note that colons replace slashes to indicate subdirectories.)

```
rez -i /:mac:lib:rincludes
```

`-o` *output-file*
Places the output in the file specified by *output-file*. Specifies the name of the associated data file; `rez` automatically appends a percent sign (`%`) to the name of the header file containing the resources. The default output file is `rez.Out`.

`-ov`
: Overrides the protection bit when resources are replaced and the `-a` option is in effect.

`-p` Writes version and progress information to diagnostic output.

`-rd`
: Suppresses warning messages if a resource type is redeclared.

resource-description-file
: Specifies files containing (source code) resource descriptions. If you don't specify any filenames, `rez` accepts keyboard input.

`-ro`
: Sets the `mapReadOnly` flag in the resource map.

`-s` *res-include-dir*
: Searches the specified directories for resource include files. (Also see the description of the `-i` option earlier in this list.)

`-t` *type*
: Specifies the type attribute of the compiled resource file. The default value is `APPL`.

`-u[ndef]` *macro*
: Undefines the macro variable *macro*. Using this option is equivalent to entering the following preprocessor request:

`#undef` *macro*

It is meaningful to undefine only the preset macro variables. (See Appendix C of ``A/UX Toolbox: Macintosh ROM Interface'' for a description of macro variables.)

DESCRIPTION

`rez` creates a resource file according to a textual series of statements in the resource description language developed for Macintosh resources. The resource description language is described in Appendix C, "Resource Compiler and Decompiler," of ``A/UX Toolbox: Macintosh ROM Interface''.

The data used to build the resource file can come directly from one or more resource description files, from other text files (through `#include` and `read` directives in the resource description file), and from other resource files (through the `include` directive in the resource-description file). The type declarations for standard Macintosh resources are contained in the files `types.r` and `systypes.r`, located in the directory `/mac/lib/rincludes`.

The `rez` command includes macro processing, full expression evaluation, built-in functions, and system variables.

The `rez` command never sends output to standard output. By default, `rez` writes to a file named `rez.Out` in the current directory. You can use the `-o` option to specify a different output file.

> *Note:* The `rez` command overwrites any existing resource of the same type and ID without a warning message. Also, `rez` cannot append resources to a resource file in which the `Read Only` bit is set. Finally, `rez` cannot replace a resource file that has a protected bit set. See also the `-ov` option in this section.

EXAMPLES

The following example is based on the descriptions in `sample.r` and the include files in the directory `/mac/lib/rincludes`. It generates a resource file for `%sample` and places the output in an AppleDouble header file named `sample`:

```
rez -i /:mac:lib:rincludes sample.r -o sample
```

STATUS MESSAGES AND VALUES

If no errors or warnings are detected, `rez` runs silently. Errors and warnings are written to standard error.

The `rez` command returns one of these status values:

0	No errors
1	Error in parameters
2	Syntax error in file
3	I/O or program error

LIMITATIONS

This command is not supported in 24-bit mode. You must run `rez` from the command line while logged in with a 32-bit Macintosh session type.

FILES

`/mac/bin/rez`
  Executable file

SEE ALSO

`derez`(1), `setfile`(1)

NAME

`rlog` — displays log messages and other information about RCS files

SYNOPSIS

`rlog` [`-d`*dates*] [`-h`] [`-l`[*lockers*]] [`-L`] [`-r`*revisions*] [`-R`] [`-s`*states*] [`-t`] [`-w`[*logins*]] *file...*

ARGUMENTS

`-d`*dates*
: Prints information about revisions with a checkin date and time in the ranges given by the semicolon-separated list of *dates.* A range of the form *d1*<*d2* or *d2*>*d1* selects the revisions that were deposited between *d1* and *d2* (inclusive). A range of the form <*d* or *d*> selects all revisions dated *d* or earlier. A range of the form *d*< or >*d* selects all revisions dated *d* or later. A range of the form *d* selects the single, latest revision dated *d* or earlier. The date and time strings *d*, *d1*, and *d2* are in the free format explained in `co(1)`. Quoting is normally necessary, especially for < and >. Note that the separator is a semicolon.

`-h` Prints only RCS filename, working filename, head, access list, locks, symbolic names, and suffix.

`-l`[*lockers*]
: Prints information about locked revisions. If the comma-separated list *lockers* of login names is given, only the revisions locked by the given login names are displayed. If the list is omitted, all locked revisions are displayed.

`-L` Ignores RCS files that have no locks set; convenient in combination with `-R`, `-h`, or `-l`.

`-r`*revisions*
: Prints information about revisions given in the comma-separated list *revisions* of revisions and ranges. A range *rev1*`-`*rev2* means revisions *rev1* to *rev2* on the same branch, `-`*rev* means revisions from the beginning of the branch up to and including *rev*, and *rev*`-` means revisions starting with *rev* to the end of the branch containing *rev*. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range.

`-R` Prints only the name of the RCS file; convenient for translating a working filename into an RCS filename.

`-s`*states*]
: Prints information about revisions whose state attributes match one of the states given in the comma-separated list *states.*

`-t` Prints the same as `-h`, plus the descriptive text.

`-w`[*logins*]
Prints information about revisions checked in by users with login names appearing in the comma-separated list *logins*. If *logins* is omitted, the user's login is assumed.

file...
Specifies the RCS files for which you want the log messages and other information.

DESCRIPTION

`rlog` displays information about RCS files. Files ending in `,v` are RCS files, all others are working files. If a working file is given, `rlog` tries to find the corresponding RCS file, first in directory `./RCS` and then in the current directory, as explained in `co`(1).

The `rlog` program displays the following information for each RCS file: RCS filename, working filename, head (that is, the number of the latest revision on the trunk), access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for display, and descriptive text. This is followed by entries for the selected revisions in reverse chronological order for each branch. For each revision, `rlog` displays revision number, author, date and time, state, number of lines added or deleted (with respect to the previous revision), locker of the revision (if any), and log message. Without options, `rlog` displays complete information; the options restrict this output.

The `rlog` command displays the intersection of the revisions selected with the `-d`, `-l`, `-s`, and `-w` options intersected with the union of the revisions selected by the `-b` and `-r` options.

EXAMPLES

The following are some examples of using `rlog`.

```
rlog -L -R RCS/*,v
rlog -L -h RCS/*,v
rlog -L -l RCS/*,v
rlog RCS/*,v
```

The first command displays the names of all RCS files in the subdirectory `RCS` which have locks. The second command displays the headers of those files, and the third displays the headers plus the log messages of the locked revisions. The last command displays complete information.

STATUS MESSAGES AND VALUES

The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 if otherwise.

NOTES

This reference manual entry describes a utility that Apple understands to have been released into the public domain by its author or authors. Apple has included this public domain utility for your convenience. Use it at your own discretion. Often the source code can be obtained if additional requirements are met, such as the purchase of a site license from an author or institution.

Author: Walter F. Tichy, Purdue University, West Lafayette, IN 47907. Copyright © 1982 by Walter F. Tichy.

SEE ALSO

`ci`(1), `co`(1), `ident`(1), `rcs`(1), `rcsdiff`(1), `rcsintro`(1), `rcsmerge`(1)

`sccstorcs`(1M) in *A/UX System Administrator's Reference*

`rcsfile`(4) in *A/UX Programmer's Reference*

Walter F. Tichy, ''Design, Implementation, and Evaluation of a Revision Control System,'' in *Proceedings of the 6th International Conference on Software Engineering,* IEEE, Tokyo, Sept. 1982

NAME

`rlogin` — logs in to a remote system

SYNOPSIS

`rlogin` *rhost* [`-8`] [`-e`*c*] [`-l` *username*]

ARGUMENTS

`-8` Allows an eight-bit data path; otherwise parity bits are stripped.

`-e`*c* Defines the following character, *c*, as the escape character. No space separates the option and the argument character.

`-l` *username*
Allows you to log into the remote system as a specified user, *username.*

DESCRIPTION

`rlogin` connects your terminal on the current local host system *lhost* to the remote host system *rhost* via a local network. On the remote side, you get whatever shell is set up for that account.

Each host has a file `/etc/hosts.equiv` which contains a list of *rhost*s with which it shares account names. (The hosts names must be the standard names as described in `remsh`(1N).) When you `rlogin` as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file `.rhosts` in his login directory. Each line in this file should contain a *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in `login`(1). To avoid security problems, the `.rhosts` file must be owned by either the remote user or root. Note that, for security reasons, root is an exception to the above; a superuser on an equivalent host must still supply the password to login as root unless the root account has its own private equivalence list in a file `.rhosts` in the root directory. Note that a `.rhosts` file for a root account is not recommended where secure systems are required.

Your remote terminal type is the same as your local terminal type (as given in your environment `TERM` variable). All echoing takes place at the remote site, so that (except for delays) the `rlogin` is transparent. Flow control via CONTROL-S and CONTROL-Q and flushing of input and output on interrupts are handled properly.

Tilde (~) is the default escape character. A line of the form "`~.`" (where "`~`" is the escape character), disconnects the current job from the remote host.

The escape sequence "~CONTROL-Z" stops the `rlogin` process and returns control to the local machine where the `rlogin` was initiated. This applies only if the initiating shell allows job control (`csh`(1) or `ksh`(1)). If your terminal suspend character (see `stty`(1)) is not CONTROL-Z, substitute that character for CONTROL-Z where applicable.

Another function of the escape character applies to nested `rlogins`. With multiple levels of `rlogins`, escapes can be sent to a specified level. When performing `rlogins` inside other `rlogins`, an escape (~CONTROL-Z) returns control to the original shell from which the first `rlogin` was initiated. However, control can be returned to other `rlogin` processes in the middle by varying the number of tildes.

If you supply two tildes, the `rlogin` shell invoked by the first `rlogin` becomes active. If you supply three tildes, the `rlogin` shell invoked by the `rlogin` shell invoked by the first `rlogin` becomes active, and so on.

For example, if you begin on machine A, then `rlogin` to machine B, then `rlogin` to machine C, "~CONTROL-Z" returns you to machine A; "~~CONTROL-z" returns you to machine B, etc.

If you choose to redefine the escape character, make sure you remember which character you have chosen, especially when nesting your `rlogins`. The escape sequence "XCONTROL-Z" returns you to the first shell that invoked an `rlogin` with "X" as the escape character. "XXCONTROL-Z" returns you to the second. So, if you `rlogin` from A to B using "X," `rlogin` from B to C using "Y," and from C to D using "X," you can get to C by typing "XXCONTROL-Z".

The second form of this command requires some preparation before it will work. The system administrator must pave the way by creating a directory, usually `/usr/hosts` and executing the following command:

```
ln /usr/bin/remsh rhost
```

which links the `remsh` binaries to *rhost*. This works for both `rlogin` and `remsh`, since the `remsh` command without a command argument is the equivalent of the `rlogin` command.

> *Note:* You must then include `/usr/hosts` (or the directory chosen by your system administrator) in the search path specified in your `.login` or `.profile` in order for the second form of this command to work.

LIMITATIONS

More terminal characteristics should be propagated.

FILES

`/usr/bin/rlogin`
: Executable file

`/usr/hosts/*`
: Directory for *rhost* version of the command

SEE ALSO

remsh(1N), stty(1)

A/UX Networking Essentials

NAME

`rm`, `rmdir` — remove files or directories

SYNOPSIS

`rm [-f] [-i] [-r]` *file...*

`rmdir` *dir...*

ARGUMENTS

dir	Specifies the directory to be removed.
`-f`	Forces the execution of the command, no questions asked. This option can be used when the standard input is not a terminal. Also, this option prevents error messages from being printed.
file	Specifies the file to be removed.
`-i`	Asks (interactively) whether to delete each file, and, under `-r`, whether to examine each directory.
`-r`	Causes an error comment to be printed unless this option is given, if the designated file is a directory. In that case, `rm` recursively deletes the entire contents of the specified directory, and the directory itself.

DESCRIPTION

`rm` removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with `y`, the file is deleted, otherwise the file remains.

`rmdir` removes entries for the named directories, which must be empty.

EXAMPLES

The command:

```
rm -rf dirname
```

will remove the entire contents of the named directory and all subdirectories, and finally the directory itself, with no questions asked.

STATUS MESSAGES AND VALUES

Generally self-explanatory. It is forbidden to remove the file `..` merely to avoid the antisocial consequences of inadvertently doing something like:

```
rm -r .*
```

FILES

`/bin/rm`
: Executable file

`/bin/rmdir`
: Executable file

SEE ALSO

`mkdir`(1)

`unlink`(2) in *A/UX Programmer's Reference*

NAME

`rmail` — handles remote mail received via UUCP

SYNOPSIS

`rmail` [`-D`*domain-name*] [`-T`] *login-name* ...

ARGUMENTS

login-name
: Specifies the login name of the user to whom the mail message is to be delivered.

`-T` Enables debugging.

`-D`*domain-name*
: Sets the domain name. The default domain name is `.UUCP`.

DESCRIPTION

`rmail` is invoked by `uuxqt` to process mail messages received by the UUCP system. The `rmail` command collapses lines that begin with `From`, as generated by mailers, such as `mail` and `mailx`, into a single line of the form *return-path*`!`*sender*, and then passes the processed message to `sendmail`. The `rmail` command is designed explicitly for use with UUCP and `sendmail` and fails if it cannot find a line that begins with `From` in the mail message.

NOTES

Mail received from a remote computer should have at least one line that begins with `From`. For example, if a mail message originates from `venus!root` and is sent to `saturn!john` via a computer named `jupiter`, the final mail message will look like this:

```
From jupiter!uucp [optional information]
From venus!root [optional information]
... text of message ...
```

As the mail message passes from computer to computer, another `From` line is added at the top until the message reaches its destination. Each computer may append optional information to the `From` line that it adds. The `rmail` command uses the `From` lines to construct a UUCP path to the user who sent the mail message. In the above example, the path would be

```
jupiter!venus!root
```

When `rmail` calls `sendmail` to deliver a message, it does so with these arguments:

`-oee`
: Causes `sendmail` to mail back error messages and to exit with 0.

`-oi`
: Causes `sendmail` to ignore periods (`.`) in the mail message.

`-oMs`*system-name* `.` *domain-name*
: Sets the `$s` macro in the line that begins with `From`. For the example given above, the value of *system-name* `.` *domain-name* would be `jupiter.UUCP`

`-oMr`*domain-name*
: Sets the `$r` macro in the domain string. For the example given above, the value of *domain-name* would be `UUCP`.

`-f`*return-path* `!` *sender*
: Specifies the return path of the user who sent the mail. For the example given above, the value of *return-path* would be `jupiter!venus` and the value of *sender* would be `root`.

login-name
: Specifies the user to whom the mail is to be delivered. For the example given above, the value of *login-name* would be `john`.

In addition, `rmail` invokes `sendmail` with the `-odq` option (queue the mail) if the load average is over 3.0. Otherwise, `rmail` invokes `sendmail` with the `-odi` option (deliver the mail immediately). The `rmail` command runs setgid on the group `sys` so it can read the `/dev/kmem` file to determine the load average.

FILES

`/bin/rmail`
: Executable file

`/usr/lib/sendmail`
: Executable file invoked by `rmail` to deliver the mail

SEE ALSO

`mail(1)`, `uucp(1C)`, `sendmail(1M)`

Chapter 9, ''Setting Up Network Mail'' in *A/UX Network System Administration*

NAME

`rmdel` — removes a delta from an SCCS file

SYNOPSIS

`rmdel -r` *SID* [*file*]...

ARGUMENTS

file Specifies the SCCS file that will be affected.

`-r` Removes the delta from the specified SCCS file.

SID Specifies the SCCS Identification String (SID).

DESCRIPTION

`rmdel` removes the delta specified by the SCCS Identification string (SID) from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the *SID* specified must not be that of a version being edited for the purpose of making a delta (i.e., if a `p-file` (see `get`(1)) exists for the named SCCS file, the SID specified must not appear in any entry of the `p-file`).

If a directory is named, `rmdel` behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with `s.`) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the ''SCCS Reference'' in *A/UX Programming Languages and Tools, Volume 2.* Simply stated, they are either: (1) if you make a delta you may remove it; or (2) if you own the file and directory, you may remove a delta.

EXAMPLES

The command:

```
rmdel -r1.2 s.test1.c
```

would remove the latest delta version (i.e., 1.2) for `s.test1.c`.

STATUS MESSAGES AND VALUES

Use `help` for explanations.

FILES

`/usr/bin/rmdel`
  Executable file
`x.file`
  Executable file
`z.file`
  Executable file

SEE ALSO

admin(1), cdc(1), comb(1), delta(1), get(1), help(1), prs(1), sact(1), sccsdiff(1), unget(1), val(1), what(1)

sccsfile(4) in *A/UX Programmer's Reference*

''SCCS Reference'' in *A/UX Programming Languages and Tools, Volume 2*

See rm(1)

NAME

`roffbib` — prints out all records in a bibliographic database

SYNOPSIS

`roffbib` [`-e`] [`-h`] [`-m` *name*] [`-n`*start-no*] [`-o`*page-range*] `-r`*letter*[*integer*] [`-s`*N*] [`-T`*tty-type*] [`-x`] [*file*]...

ARGUMENTS

`-e` Produces equally spaced words in adjusted lines, using the full resolution of the particular terminal.

file Specifies the file into which the output from the `roffbib` command is placed.

`-h` Uses output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

`-m`*name*
: Prepends to the input *file*s to the `/usr/lib/tmac/tmac.`*name* macro file.

`-n`*start-no*
: Numbers the first generated page at *start-no*.

`-o`*page-range*
: Prints only pages whose page numbers appear in the *page-range* of numbers and ranges, separated by commas. A range *x*–*y* means pages *x* through *y*; a range given by –*y* means from the beginning to page *y*; a range given by *x*– means from page *x* to the end.

`-r`*letter*[*integer*]
: Set the number register referenced by *letter* to *integer*.

`-s`[*pages-per-pause*]
: Specifies the number of pages to print between pauses, causing `nroff to halt to allow paper` receipt of a linefeed or newline (newlines do not work in pipelines, e.g., with `mm`). The default is 1. This option does not work if the output of `nroff` is piped through `col`. When `nroff` (`otroff`) halts between pages, an ASCII `BEL` is sent to the terminal.

`-T`*tty-type*
: Prepare output for specified terminal. Known *tty-type*s are

`2631`	Hewlett-Packard 2631 printer in regular mode
`2631-c`	Hewlett-Packard 2631 printer in compressed mode
`2631-e`	Hewlett-Packard 2631 printer in expanded mode

`300`	DASI-300 printer
`300-12`	DASI-300 terminal set to 12-pitch (12 characters per inch)
`300s`	DASI-300s printer (300s is a synonym)
`300s-12`	DASI-300s terminal set to 12-pitch (12 characters per inch) (300s-12 is a synonym)
`37`	TELETYPE Model 37 terminal (default)

`-x` Suppresses the printing of these abstracts.

DESCRIPTION

`roffbib` prints out all records in a bibliographic database, in bibliography format rather than as footnotes or endnotes. Generally it is used in conjunction with `sortbib`:

`sortbib` *database* `| roffbib`

The `roffbib` program accepts most of the options understood by `nroff`; most importantly, `-T` which specifies terminal type.

If abstracts or comments are entered following the `%X` field key, `roffbib` will format them into paragraphs for an annotated bibliography. Several `%X` fields may be given if several annotation paragraphs are desired.

Four command-line registers control formatting style of the bibliography, much like the number registers of `ms`. The command-line argument `-rN1` will number the references starting at one (1). The flag `-rV2` will double space the bibliography, while `-rV1` will double space references but single space annotation paragraphs. The line length can be changed from the default 6.5 inches to 6 inches with the `-rL6i` argument, and the page offset can be set from the default of 0 to one inch by specifying `-rO1i` (capital O, not zero).

LIMITATIONS

Users have to rewrite macros to create customized formats.

FILES

`/usr/ucb/roffbib`
    Executable file

`/usr/lib/tmac/tmac.bib`
    Macro file

SEE ALSO

`addbib`(1), `indxbib`(1), `lookbib`(1), `nroff`(1), `refer`(1), `sortbib`(1)

NAME

`rpcgen` — generates C source code from a remote procedure call (RPC) source file

SYNOPSIS

`rpcgen` *input-file*

`rpcgen -c [-o` *output-file*`]` [*input-file*]

`rpcgen -h [-o` *output-file*`]` [*input-file*]

`rpcgen -l [-o` *output-file*`]` [*input-file*]

`rpcgen -m [-o` *output-file*`]` [*input-file*]

`rpcgen -s` *transport* `[-o` *output-file*`]` [*input-file*]

ARGUMENTS

`-c` Causes `rpcgen` to produce only external data representation (XDR) routines.

`-h` Causes `rpcgen` to produce only C data definitions. The output is suitable for use as a C include file.

input-file
Specifies the name of a file containing RPC source code. The filename must have a suffix of `.x`. The `rpcgen` command uses the value of *input-file* to form the filenames of the output files. For example, if the value of *input-file* is `proto.x`, `rpcgen` produces a C include file in `proto.h`, XDR routines in `proto_xdr.c`, server-side stubs in `proto_svc.c`, and client-side stubs in `proto_clnt.c`.

`-l` Causes `rpcgen` to produce only client-side stubs.

`-m` Causes `rpcgen` to produce only server-side stubs, but not to generate a routine called `main`. This option is useful when you are writing callback routines and when you need to write your own `main` routine to do initialization.

`-o` *output-file*
Specifies the name of the output file. If you do not use this option, `rpcgen` writes to the standard output. You can use this option only when you also use a `-c`, `-h`, `-l`, `-m`, or `-s` option.

`-s` *transport*
Causes `rpcgen` to produce server-side stubs, using the transport specified by *transport*. The value of *transport* can be `udp` or `tcp`. You can use this option more than once to produce an output that supports multiple transports.

DESCRIPTION

`rpcgen` generates C code that implements a RPC protocol. The input to `rpcgen` is written in the RPC language, which is similar to the C language. See *A/UX Network Programming Applications* for information about the syntax of the RPC language.

The `rpcgen` program is most commonly run with just an *input-file* argument. In this case, `rpcgen` generates four output files. There is one output file for each of these output types: C data definitions, XDR routines, server-side stubs, and client-side stubs. You can use an option to generate only one of the output types.

The `rpcgen` command automatically runs `cpp` on all input files before they are interpreted, so all legal `cpp` directives are legal within an `rpcgen` input file. For each type of output file, `rpcgen` defines one of these special `cpp` symbols that you can use in your source code:

`RPC_CLNT`
: This symbol is defined when you are producting client-side stubs.

`RPC_HDR`
: This symbol is defined when you are producing a C include file.

`RPC_SVC`
: This symbol is defined when you are producing server-side stubs.

`RPC_XDR`
: This symbol is defined when you are producing XDR routines.

The `rpcgen` command passes any line beginning with a percent sign (`%`) into the output file without interpreting it.

You can customize some of your XDR routines by leaving those data types undefined. For every data type that is undefined, `rpcgen` assumes that there exists a routine with the name `xdr_` prepended to the name of the undefined type.

LIMITATIONS

Nesting is not supported. To achieve the same effect, you can declare structures at the top level and use their names inside other structures.

Name clashes can occur when you are using program definitions because the apparent scoping does not really apply. You can avoid most clashes by giving unique names for programs, versions, procedures, and types.

SEE ALSO

`cpp(1)`

''Remote Procedure Call (RPC) Programming Guide'' in *A/UX Network Applications Programming* (which is available from APDA)

See sh(1)

NAME

`rup` — displays the status of machines on the local network (RPC version)

SYNOPSIS

`rup` [`-h`] [`-l`] [`-t`] [*host*]...

ARGUMENTS

`-h`	Sorts the display alphabetically by host name.
`-l`	Sorts the display by load average.
`-t`	Sorts the display by up time.
host	Specifies the host whose status is to be displayed.

DESCRIPTION

`rup` gives a status similar to `uptime` for remote machines; it broadcasts on the local network, and displays the responses it receives.

Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options.

When *host* arguments are given, rather than broadcasting, `rup` will only query the list of specified hosts.

A remote host will only respond if it is running the `rstatd` daemon, which is normally started up from `inetd`(1M).

LIMITATIONS

Broadcasting does not work through gateways.

FILES

`/usr/bin/rup`
Executable file

`/etc/servers`
File containing list of hosts

SEE ALSO

`ruptime`(1N)

`inetd`(1M), `rstatd`(1M) in *A/UX System Administrator's Reference*

NAME

ruptime — displays the host status of local machines

SYNOPSIS

`ruptime [-a] [-l] [-t] [-u]`

ARGUMENTS

- `-a` Displays users who are idle an hour or more on the system in addition to the normal display.
- `-l` Sorts the listing by load average.
- `-t` Sorts the listing by uptime.
- `-u` Sorts the listing by number of users.

DESCRIPTION

`ruptime` gives a status line like `uptime` for each machine on some local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 5 minutes are shown as being down.

Normally, the listing is sorted by host name.

FILES

`/usr/bin/ruptime`
Executable file

`/usr/spool/rwho/whod.*`
File containing system information

SEE ALSO

rwho(1N), uptime(1).

NAME

`rusers` — produces a login list for local machines (RPC version)

SYNOPSIS

`rusers` [`-a`] [`-h`] [`-i`] [`-l`] [`-u`] [*host*]...

ARGUMENTS

`-a` Gives a report for a machine even if no users are logged on.

`-h` Sorts alphabetically by host name.

host Queries only the list of specified hosts, rather than broadcasting to the entire network.

`-i` Sorts by idle time.

`-l` Gives a longer listing in the style of `who`(1). In addition, if a user hasn't typed to the system for a minute or more, the idle time is reported.

`-u` Sorts by number of users.

DESCRIPTION

The `rusers` command produces output similar to `users`(1) and `who`(1), but for remote machines. It broadcasts on the local network and prints the responses it receives. Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed later.

The default is to print out a listing in the style of `users`(1) with one line per machine.

A remote host will respond only if it is running the `rusersd` daemon, which is normally started up from `inetd`.

FILES

`/usr/etc/rusers`
Executable file

`/etc/servers`
File containing list of servers

SEE ALSO

`rwho`(1N), `inetd`(1M), `rusersd`(1M), `servers`(4).

NAME

rwho — displays a list of the active users from all of the systems on the local network

SYNOPSIS

`rwho [-a]`

ARGUMENTS

`-a` Causes `rwho` to display information about the users who have not typed on the system for an hour or more

DESCRIPTION

The `rwho` command produces output similar to `who`, but for all machines on the local network. If no report has been received from a machine for 5 minutes then `rwho` assumes the machine is down, and does not report users last known to be logged into that machine.

If a user hasn't typed to the system for a minute or more, then `rwho` reports this idle time. If a user hasn't typed to the system for an hour or more, then the user will be omitted from the output of `rwho` unless the `-a` option is given.

LIMITATIONS

This is unwieldy when the number of machines on the local net is large.

FILES

`/usr/bin/rwho`
Executable file

`/usr/spool/rwho/whod.*`
File containing system information

SEE ALSO

`ruptime`(1N)

`rwhod`(1M) in *A/UX System Administrator's Reference*

NAME

sact — displays who has checked a Source Code Control System (SCCS) file out for editing

SYNOPSIS

sact [-] *file...*

ARGUMENTS

-	Reads the standard input with each line being taken as the name of an SCCS file to be processed.
file	Specifies the SCCS file that is checked out.

DESCRIPTION

sact informs you of any impending deltas to a named SCCS file. This situation occurs when get with the -e option has been previously executed without a subsequent execution of delta. If a directory is named on the command line, sact behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored.

The output for each named file consists of five fields separated by spaces:

Field 1	Specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	Specifies the SID for the new delta to be created.
Field 3	Contains the login name of the user who will make the delta (that is, the user who executed a get for editing).
Field 4	Contains the date that get -e was executed.
Field 5	Contains the time that get -e was executed.

EXAMPLES

If the user has issued a get -e command, but not a delta command, to merge the new changes, issuing

```
sact s.test1.c
```

would show

```
1.2 1.3 virginia 82/11/10 16:10:35
```

indicating that a new version numbered 1.3 is in the process of being made from version numbered 1.2 by user virginia. The get -e command for the file was issued on 82/11/10 at 16:10:35.

STATUS MESSAGES AND VALUES

Use `help` for explanations.

FILES

`/usr/bin/sact`
: Executable file

SEE ALSO

`admin`(1), `cdc`(1), `comb`(1), `delta`(1), `get`(1), `help`(1), `prs`(1), `rmdel`(1), `sccs`(1), `sccsdiff`(1), `unget`(1), `val`(1), `what`(1)

`sccsfile`(4) in *A/UX Programmer's Reference*

''SCCS Reference'' in *A/UX Programming Languages and Tools, Volume 2*

NAME

`sag` — generates a system activity graph

SYNOPSIS

`sag` [`-e` *time*] [`-f` *file*] [`-i` *sec*] [`-s` *time*] [`-T` *term*] [`-x` *spec*] [`-y` *spec*]

ARGUMENTS

`-e` *time*

Selects data up to *time* in the form *hh*[:*mm*], where *hh* is the time in hours (military time) and *mm* is the time in minutes. The default is `18:00`.

`-f` *file*

Specifies the file used as the data source for `sar`. The default is the current daily data file `/usr/adm/sa/sa`*dd*.

The *file* argument specifies 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 as

```
(A/(A+B))*100
```

and

```
A + B / C + D
```

is

```
(A+B)/(C+D)
```

`-i` *sec*

Selects data at intervals as close as possible to *sec* seconds.

`-s` *time*

Selects data later than *time*. The default is `08:00`.

`-T` *term*

Produces output suitable for terminal *term*. See `tplot`(1G) for known terminals. If *term* is `vpr`, output is processed by `vpr -p` and queued to a Versatec printer. Default for *term* is `$TERM`.

`-x` *spec*

Specifies the x axis with *spec* in the form:

```
name [ op  name ]...[lo  hi ]
```

`-y` *spec*

Specifies the y axis with *spec* in the form:

```
name [ op  name ]...[lo  hi ]
```

The `-y` default is:

```
-y ''%usr 0 100; %usr + %sys 0 100; \
%usr + %sys + %wio 0 100''
```

DESCRIPTION

`sag` graphically displays the system activity data stored in a binary data file by a previous `sar` 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. `sag` invokes `sar` and finds the desired data by string-matching the data column header (run `sar` to see what's available). *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 \RETURN are included.

EXAMPLES

Entering:

```
sag
```

will show today's CPU utilization.

FILES

`/usr/bin/sag`
: Executable file

`/usr/adm/sa/sa`*dd*
: Daily data file for day *dd*

SEE ALSO

sar (1), tplot (1G)

NAME

`sar` — reports system activity

SYNOPSIS

`sar` [`-a`] [`-A`] [`-b`] [`-c`] [`-m`] [`-q`] [`-u`] [`-v`] [`-w`] [`-y`] [`-o`*file*] *t* [*n*]

`sar` [`-a`] [`-A`] [`-b`] [`-c`] [`-e`*time*] [`-f`*file*] [`-i`*sec*] [`-m`] [`-q`] [`-s`*time*] [`-u`] [`-v`] [`-w`] [`-y`]

ARGUMENTS

`-a` Reports use of file access system routines:
`iget/s`, `namei/s`, `dirblk/s`.

`-A` Reports all data. This option is equivalent to the `-u`, `-q`, `-b`, `-w`, `-c`, `-a`, `-y`, `-v`, and `-m` options.

`-b` Reports 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, for example, `1-bread/lread`.
`pread/s`, `pwrit/s`: transfers via raw (physical) device mechanism.

`-c` Reports 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.

`-e`*time*
Specifies the ending time of the report. The format for *time* is *hh*[:*mm*[:*ss*]].

`-f`*file*
Extracts data from the given previously-recorded *file*.

`-i`*sec*
Selects records at *sec* second intervals. If this option is not specified, all intervals found in the data file are reported.

`-m` Reports message and semaphore activities.
`msg/s`, `sema/s`: primitives per second.

n Specifies *n* intervals. The default value for this option is 1.

`-o`*file*
Saves the samples in *file* in binary format.

`-q` Reports average queue length while occupied and percentage 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.

`-s`*time*
: Specifies the starting time of the report. The format for *time* is *hh*[:*mm*[:*ss*]].

t Specifies *t* seconds.

`-u` Reports 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.

`-v` Reports status of process, inode, file, file record lock and file record header tables.
`proc-sz`, `inod-sz`, `file-sz`, `lock-sz`, `fhdr-sz`: entries/size for each table, evaluated once at sampling point;
`proc-ov`, `inod-ov`, `file-ov`: overflows occurring between sampling points.

`-w` Reports system swapping and switching activity:
`swpin/s`, `swpot/s`, `bswin/s`, `bswot/s`: `bswin/s`, `bswot/s`: number of transfers and number of 512 byte units transferred for swapins (including initial loading of some programs) and swapouts;
`pswch/s`: process switches.

`-y` Reports TTY device activity:
`rawch/s`, `canch/s`, `outch/s`: input character rate, input character rate processed by canon, and output character rate.
`rcvin/s`, `xmtin/s`, `mdmin/s`: receive, transmit and modem interrupt rates.

DESCRIPTION

`sar`, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds.

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*.

EXAMPLES

The command:

```
sar
```

shows today's CPU activity so far. The command:

```
    sar -o temp 60 10
```

watches CPU activity evolve for 10 minutes and saves data.

FILES

`/usr/bin/sar`
: Executable file

`/usr/adm/sa/sa`*day-of-month*
: Daily data file, where *day-of-month* are digits representing the day of the month

SEE ALSO

`sag`(1G)

`sadc`(1M) in *A/UX System Administrator's Reference*

''System Activity Package'' in *A/UX Local System Administration*

NAME

`sccs` — performs SCCS subsystem commands

SYNOPSIS

`sccs` *command* [*flags*] [*args*] [`-d`*path*] [`-p`*path*] [`-r`]

ARGUMENTS

args Specifies the arguments to *command*.

`-d`*path* Gives a root directory, *path*, for the SCCS files. The default is the current directory.

flags Specifies the data to be interpreted by the `sccs` program, or the data to be passed to the actual SCCS program. Any *flag* options to be interpreted by the `sccs` program must appear before *command*; any *flags* to be passed to the actual SCCS program must come after *command*. These flags are specific to the command and are discussed in the documentation for that command.

`-p`*path* Defines the pathname of the directory, *path*, in which the SCCS files will be found; SCCS is the default. This option differs from the `-d` option in that the `-d` option is prefixed to the entire pathname and the `-p` argument is inserted before the final component of the pathname. For example:

```
sccs -d /x -py get a/b
```

will convert to

```
get /x/a/y/s.b
```

The intent here is to create aliases such as:

```
alias syssccs sccs -d /usr/src
```

which will be used as:

```
syssccs get cmd/who.c
```

Also, if the environment variable `PROJECT` is set, its value is used to determine the `-d` option. If it begins with a slash, it is taken directly; otherwise, the home directory of a user of that name is examined for a subdirectory `src` or `source`. If such a directory is found, it is used.

`-r` Runs `sccs` as the real user rather than as whatever effective user `sccs` has set user ID to.

DESCRIPTION

`sccs` is a front end to the SCCS programs that helps them mesh more cleanly with the rest of A/UX. It also includes the capability to run set-user-id to another user to provide additional protection.

Basically, `sccs` runs the given *command* with the specified *flags* and *args*. Each argument is normally modified to be prefixed with `SCCS/s`. Thus, you may supply `get`, `delta`, or `info` in place of *command*.

Besides the usual SCCS commands, several *pseudo-commands* can be issued, which are:

`Cedit`	Equivalent to `get -e`.
`delget`	Perform a delta on the named files and then get new versions. The new versions will have ID keywords expanded, and will not be editable. The `-m`, `-p`, `-r`, `-s`, , and `-y` options will be passed to `delta`, and the `-b`, `-c`, `-e`, `-i`, `-k`, `-l`, `-s`, and `-x` options will be passed to `get`.
`deledit`	Equivalent to `delget`, except that the `get` phase includes the `-e` option. This option is useful for making a ''checkpoint'' of your current editing phase. The same options will be passed to `delta` as described above, and all the options listed for `get` above except `-e`and`-k` are passed to `edit`.
`create`	Create an SCCS file, taking the initial contents from the file of the same name. Any options to `admin` are accepted. If the creation is successful, the files are renamed with a comma on the front. These should be removed when you are convinced that the SCCS files have been created successfully.
`fix`	Must be followed by a `-r` option. This command essentially removes the named delta, but leaves you with a copy of the delta with the changes that were in it. It is useful for fixing small compiler bugs, etc. Since it doesn't leave audit trails, it should be used carefully.
`clean`	This routine removes everything from the current directory that can be recreated from SCCS files. It will not remove any files being edited. If the `-b` option is given, branches are ignored in the determination of whether they are being edited; this is dangerous if you are keeping the branches in the same directory.
`unedit`	This is the opposite of an `edit` or a `get -e`. It should be used with extreme caution, since any changes you made since the `get` will be irretrievably lost.
`info`	Give a listing of all files being edited. If the `-b` option is given, branches (i.e., SID's with two or fewer components) are ignored. If the `-u` option is given (with an optional argument) then only files being edited by you (or the named user) are listed.

`check`	Like `info`, except that nothing is printed if nothing is being edited, and a nonzero exit status is returned if anything is being edited. The intent is to have this included in an `install` entry in a makefile to ensure that everything is included into the SCCS file before a version is installed.
`tell`	Give a newline-separated list of the files being edited on the standard output. Takes the `-b` and `-u` options like `info` and `check`.
`diffs`	Give a `diff` listing between the current version of the program(s) you have out for editing and the versions in SCCS format. The `-r`, `-c`, `-i`, `-x`, and `-t` options are passed to `get`; the `-l`, `-s`, `-e`, `-f`, `-h`, and `-b` options are passed to `diff`. The `-C` option is passed to `diff` as `-c`.
`prs`	This command prints out verbose information about the named files.

Certain commands (such as `admin`) cannot be run `set user id` by all users, since this would allow anyone to change the authorizations. These commands are always run as the real user.

EXAMPLES

To get a file for editing, edit it, and produce a new delta:

```
sccs get -e file.c
ex file.c
sccs delta file.c
```

To get a file from another directory:

```
sccs -p/usr/src/sccs/s. get cc.c
```

or

```
sccs get /usr/src/sccs/s.cc.c
```

To make a delta of a large number of files in the current directory:

```
sccs delta *.c
```

To get a list of files being edited that are not on branches:

```
sccs info -b
```

To delta everything being edited by you:

```
sccs delta 'sccs tell -u'
```

In a *makefile*, to get source files from an SCCS file if it does not already exist:

```
SRCS = <list of source files>
$(SRCS):
```

```
            sccs get $(REL) $@
```

LIMITATIONS

It should be able to take directory arguments on pseudo-commands like the SCCS commands do.

FILES

`/usr/ucb/sccs`
    Executable file

SEE ALSO

`admin`(1), `cdc`(1), `comb`(1), `delta`(1), `get`(1), `help`(1), `prs`(1), `rmdel`(1), `sact`(1), `sccsdiff`(1), `unget`(1), `val`(1), `what`(1)

`sccsfile`(4) in *A/UX Programmer's Reference*

''SCCS Reference'' in *A/UX Programming Languages and Tools, Volume 2*

NAME

`sccsdiff` — compares two versions of an SCCS file

SYNOPSIS

`sccsdiff -r`*delta1* `-r`*delta2* [`-p`] [`-s`*n*] *file...*

ARGUMENTS

file	Specifies the SCCS file to be compared.
`-p`	Pipes output for each file through `pr`.
`-r`*delta1*	Specifies the earlier delta version of an SCCS file This file is to be compared with *delta2*.
`-r`*delta2*	Specifies the later delta version of an SCCS file This file is to be compared with *delta2*.
`-s`*n*	Specifies the file segment size that is passed to `diff`. This is useful when `diff` fails due to a high system load.

DESCRIPTION

`sccsdiff` compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files. `sccsdiff` first outputs lines resembling the `ed`(1) commands to convert *file1* into *file2*. It then outputs the actual lines that differ.

EXAMPLES

To show the differences between version 1.1 and version 1.2 of the file, `test1.c` enter:

```
sccsdiff -r1.1 -r1.2 s.test1.c
```

STATUS MESSAGES AND VALUES

file: `No differences`
If the two versions are the same.

Use `help` for explanations.

FILES

`/usr/bin/sccsdiff`
Executable file

`/tmp/get?????`
Temporary file

SEE ALSO

`admin`(1), `bdiff`(1), `cdc`(1), `comb`(1), `delta`(1), `diff`(1), `get`(1), `help`(1), `pr`(1), `sccs`(1)

"SCCS Reference" in *A/UX Programming Languages and Tools, Volume 2*

NAME

`script` — starts a shell that records terminal input and output

SYNOPSIS

`script [-a]` [*file*]

ARGUMENTS

`-a` Writes or appends the typescript to *file.*

file Specifies the file that is to be written to. If a filename is not given, the typescript is saved in the file `typescript`.

DESCRIPTION

`script` makes a typescript of everything printed on your terminal. The typescript is written to *file*, or appended to *file* if the `-a` option is given. It can be sent to the line printer later with the `lp` or `lpr` commands.

Note that `script` uses both standard input and standard output and that neither may be redirected via a pipe, < or >.

The script ends when the forked shell exits.

This program is useful when using a CRT and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply.

LIMITATIONS

`script` places everything in the log file, meaning everything typed or appearing on the screen, including control characters. If `vi` is invoked, whatever appeared on-screen (including invisible characters) will be placed in the log file. Control characters useful for screen output will appear as garbage and will be illegible in a script. Thus, it is a good idea not to use `vi` while using `script`.

FILES

`/usr/bin/script`
Executable file

SEE ALSO

`lpr`(1)

NAME

`sdb` — symbolic debugger

SYNOPSIS

`sdb` [`-w`] [`-W`] [*objfile* [*corfile* [*directory*]]]

ARGUMENTS

corfile
: Specifies a core image file produced after executing *objfile*; the default for *corfile* is `core`. The core file need not be present. A `-` in place of *corfile* forces `sdb` to ignore any core image file.

directory
: Specifies the directory into which the source files used in constructing *objfile* must be placed.

objfile
: Specifies an executable program file which has been compiled with the `-g` (debug) option. If the file has not been compiled with the `-g` option, or if it is not an executable file, the symbolic capabilities of `sdb` are limited, but the file can still be examined and the program debugged. The default for *objfile* is `a.out`.

`-w` Permits overwriting locations in *objfile*.

`-W` Disables the checking feature and its accompanying warnings that are produced when an *objfile* is produced.

DESCRIPTION

`sdb` is a symbolic debugger which can be used with C and Fortran programs. It may be used to examine their object files and core files and to provide a controlled environment for their execution.

It is useful to know that at any time there is a ''current line'' and ''current file.'' If *corfile* exists, then they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in `main`(). The current line and file may be changed with the source file examination commands.

By default, warnings are provided if the source files used in producing *objfile* cannot be found, or are newer than *objfile*. This checking feature and the accompanying warnings may be disabled by the use of the `-W` option.

Names of variables are written just as they are in C or `f77`(1). Variables local to a procedure may be accessed using the form *procedure*:*variable*. If no procedure name is given, the procedure containing the current line is used by default.

It is also possible to refer to structure members as *variable . member ,* pointers to structure members as *variable->member ,* and array elements as *variable*[number]. Pointers may be dereferenced by using the form *pointer*[0] . Combinations of these forms may also be used. f77 common variables may be referenced by using the name of the common block instead of the structure name. Blank common variables may be named by the form *. variable .* A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by sdb. An unqualified structure variable may also be used with various commands. Generally, sdb interprets a structure as a set of variables; thus, it displays the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value sdb displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as

variable[number] [number]...

or as

variable[number,number,...]

In place of *number*, the form *number ; number* may be used to indicate a range of values, * may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, sdb displays all the values of an array or of the section of an array if trailing subscripts are omitted. It displays only the address of the array itself or of the section specified by the user if subscripts are omitted. A multidimensional parameter in an f77 program cannot be displayed as an array, but it is actually a pointer, whose value is the location of the array. The array itself can be accessed symbolically from the calling function.

A particular instance of a variable on the stack may be referenced by using the form *procedure : variable , number .* All the variations mentioned in naming variables may be used. *number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C may be used, so that addresses may be input in decimal, octal, or hexadecimal.

Line numbers in the source program are referred to as *file-name* : *number* or *procedure* : *number* . In either case the number is relative to the beginning of the file. If no procedure or filename is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under `sdb` all addresses refer to the executing program; otherwise they refer to *objfile* or *corfile.*

Addresses

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1, e1, f1*) and (*b2, e2, f2*). The *file address* corresponding to a written *address* is calculated as follows:

b1address<e1

file address=address+f1-b1

otherwise

b2address<e2

file address=address+f2-b2,

otherwise, the requested *address* is not legal. In some cases (for example, for programs with separated I and D space) the two segments for a file may overlap.

The initial setting of both mappings is suitable for normal `a.out` and `core` files. If either file is not of the kind expected then, for that file, *b1* is set to 0, *e1* is set to the maximum file size, and *f1* is set to 0; in this way the whole file can be examined with no address translation.

In order for `sdb` to be used on large files, all appropriate values are kept as signed 32-bit integers.

Commands

The commands for examining data in the program are:

`t` Prints a stack trace of the terminated or halted program.

`T` Prints the top line of the stack trace.

variable / clm
: Prints the value of *variable* according to length *l* and format *m.* A numeric count *c* indicates that a region of memory, beginning at the address implied by *variable*, is to be displayed. The length specifiers are:

`b` one byte

`h`	two bytes (half word)
`l`	four bytes (long word)

Legal values for *m* are:

`c`	character
`d`	decimal
`u`	decimal, unsigned
`o`	octal
`x`	hexadecimal
`f`	32-bit single precision floating point
`g`	64-bit double precision floating point
`s`	Assumes *variable* is a string pointer and print characters starting at the address pointed to by the variable.
`a`	Prints characters starting at the variable's address. This format may not be used with register variables.
`p`	pointer to procedure
`i`	Disassembles machine-language instruction with addresses printed numerically and symbolically.
`I`	Disassembles machine-language instruction with addresses printed numerically only.

The length specifiers are only effective with the formats `c`, `d`, `u`, `o` and `x`. Any of the specifiers, *c*, *l*, and *m*, may be omitted. If all are omitted, `sdb` chooses a length and a format suitable for the variable's type, as declared in the program. If *m* is specified, then this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier *c* tells `sdb` to display that many units of memory, beginning at the address of *variable*. The number of bytes in one such unit of memory is determined by the length specifier *l*, or, if no length is given, by the size associated with the *variable*. If a count specifier is used for the `s` or `a` command, then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command `./`.

The `sh`(1) metacharacters `*` and `?` may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, variables local to the current procedure and global variables are matched; if a procedure name is specified, only variables local to that procedure are matched. To match only global variables, the form `:`*pattern* is used.

linenumber`?`*lm*
variable:`?`*lm*

Prints the value at the address from `a.out` or I space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is 'i'.

variable`=`*lm*
linenumber`=`*lm*
number`=`*lm*

Prints the address of *variable* or *linenumber*, or the value of *number*, in the format specified by *lm*. If no format is given, then `lx` is used. The last variant of this command provides a convenient way to convert between decimal, octal and hexadecimal.

variable`!`*value*

Sets *variable* to the given *value*. The value may be a number, a character constant or a variable. The value must be well defined; expressions that produce more than one value, such as structures, are not allowed. Character constants are denoted `'`*character*. Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type double. Registers are viewed as integers. The *variable* may be an expression that indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type *int*. C conventions are used in any type conversions necessary to perform the indicated assignment.

`x` Prints the machine registers and the current machine-language instruction.

`X` Prints the current machine-language instruction.

The commands for examining source files are:

`e` *procedure*
`e` *file-name*
`e` *directory/*
`e` *directory file-name*

The first two forms set the current file to the file containing *procedure* or to *file-name*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The default is the current working directory. The latter two forms change the value of *directory*. If no procedure, filename, or directory is given, the current procedure name and filename are reported.

/regular expression/

Searches forward from the current line for a line containing a string matching *regular expression* as in `ed(1)`. The trailing `/` may be elided.

`?`*regular expression*`?`
: Searches backward from the current line for a line containing a string matching *regular expression* as in `ed`(1). The trailing `?` may be elided.

`p` Prints the current line.

`z` Prints the current line followed by the next 9 lines. Set the current line to the last line printed.

`w` Creates a window. Prints the 10 lines around the current line.

number
: Sets the current line to the given line number, then prints the new current line.

count+
: Advances the current line by *count* lines, then prints the new current line.

count `-`
: Retreats the current line by *count* lines, then prints the new current line.

The commands for controlling the execution of the source program are:

count `r` *args*
count `R`
: Runs the program with the given arguments. The `r` command with no arguments reuses the previous arguments to the program while the `R` command runs the program with no arguments. An argument beginning with < or > causes redirection for the standard input or output respectively. If *count* is given, it specifies the number of breakpoints to be ignored.

linenumber `c` *count*
linenumber `C` *count*
: Continues after a breakpoint or interrupt. If *count* is given, it specifies the number of breakpoints to be ignored. `C` continues with the signal that caused the program to stop reactivated and `c` ignores it. If a linenumber is specified then a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted when the command finishes.

linenumber `g` *count*
: Continues after a breakpoint with execution resumed at the given line. If *count* is given, it specifies the number of breakpoints to be ignored.

`s` *count*
`S` *count*
: Single steps the program through *count* lines. If no count is given then the program is run for one line. `S` is equivalent to `s` except it steps through procedure calls.

`i`
`I` Single steps by one machine-language instruction. `I` steps with the signal that caused the program to stop reactivated and `i` ignores it.

variable$`m` *count*
address:`m` *count*
: Single steps (as with `s`) until the specified location is modified with a new value. If *count* is omitted, it is effectively infinity. *variable* must be accessible from the current procedure. Since this command is done by software, it can be very slow.

level `v`
: Toggles verbose mode, for use when single stepping with `S`, `s`, or `m`. If *level* is omitted, then just the current source file and/or subroutine name is printed when either changes. If *level* is 1 or greater, each C source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A `v` turns verbose mode off if it is on for any level.

`k` Kills the program being debugged.

procedure`(`*arg1*`,`*arg2*`,`*...*`)`
procedure`(`*arg1*`,`*arg2*`,`*...*`)`*/m*
: Executes the named procedure with the given arguments. Arguments can be integer, character or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format *m*. If no format is given, it defaults to `d`.

linenumber `b` *commands*
: Sets a breakpoint at the given line. If a procedure name without a line number is given (for example, `proc:`), a breakpoint is placed at the first line in the procedure even if it was not compiled with the `-g` option. If no *linenumber* is given, a breakpoint is placed at the current line. If no *commands* are given, execution stops just before the breakpoint and control is returned to `sdb`. Otherwise the *commands* are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons. If `k` is used as a command to execute at a breakpoint, control returns to `sdb`, instead of continuing execution.

`B` Prints a list of the currently active breakpoints.

linenumber `d`
Deletes a breakpoint at the given line. If no *linenumber* is given, the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from the standard input. If the line begins with a `y` or `d`, the breakpoint is deleted.

`D` Deletes all breakpoints.

`l` Prints the last executed line.

linenumber `a`
Announces. If *linenumber* is of the form *proc* `:` *number*`,` the command effectively does a *linenumber* `bl`. If *linenumber* is of the form *proc* `:`, the command effectively does a *proc* `:` `b T`.

Miscellaneous commands:

`!`*command*
Interprets the command by `sh`(1).

`newline`
Advances the current line by one line and print the new current line if the previous command printed a source line. If the previous command displayed a memory location, displays the next memory location.

`CONTROL-D`
Scrolls. Print the next 10 lines of instructions, source, or data, depending on which was printed last.

`<` *filename*
Reads commands from *filename* until the end of file is reached, then continues to accept commands from standard input. When `sdb` is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; < may not appear as a command in a file.

`M` Prints the address maps.

`M[/][*]` *b e f*
Records new values for the address map. The arguments `?` and `/` specify the text and data maps, respectively. The first segment, (*b1, e1, f1*), is changed unless `*` is specified, in which case the second segment, (*b2, e2, f2*), of the mapping is changed. If fewer than three values are given, the remaining map parameters are left unchanged.

`"` *string*
Prints the given string. The C escape sequences of the form `\`*character* are recognized, where *character* is a nonnumeric

character.

`q` Exits the debugger.

The following commands also exist and are intended only for debugging the debugger:

`V` Prints the version number.

`Q` Prints a list of procedures and files being debugged.

`Y` Toggles debug output.

WARNINGS

Data stored in text sections are indistinguishable from functions.

Line number information in optimized functions is unreliable, and some information may be missing.

LIMITATIONS

If a procedure is called when the program is not stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

The default type for printing `f77` parameters is incorrect. Their address is printed instead of their value.

Tracebacks containing `f77` subprograms with multiple entry points may print too many arguments in the wrong order, but their values are correct.

The range of an `f77` array subscript is assumed to be *1* to *n*, where *n* is the dimension corresponding to that subscript. This is only significant when the user omits a subscript, or uses `*` to indicate the full range. There is no problem in general with arrays having subscripts whose lower bounds are not 1.

FILES

`/usr/bin/sdb`
  Executable file

`a.out`
  Compiled file

`core`
  Core file

SEE ALSO

`adb`(1), `cc`(1), `ctrace`(1), `f77`(1), `sh`(1)

`a.out`(4), `core`(4) in *A/UX Programmer's Reference*

``sdb` Reference'' in *A/UX Programming Languages and Tools, Volume 1*

NAME

`sdiff` — reports side-by-side differences between two files in a side-by-side format

SYNOPSIS

`sdiff` [`-l`] [`-o` *output*] [`-s`] [`-w` *cols*] *file1* *file2*

ARGUMENTS

file1
  Specifies the file to be compared with *file2*.

file2
  Specifies the file to be compared with *file1*.

`-l` Prints only the left side of any lines that are identical.

`-o` *output*
  Specifies *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*.

`-s` Does not print identical lines.

`-w` *cols*
  Specifies *cols*, as the width of the output line. The default line length is 130 characters. The width must be between 20 and 200.

DESCRIPTION

`sdiff` uses the output of `diff` 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.

Sets of differences, as produced by `diff`(1), are printed; 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` appends the left column to the output file

`r` appends the right column to the output file

`s` turns on silent mode; does not print identical lines

`v` turns off silent mode

`e l`
  calls the editor with the left column

`e r`
  calls the editor with the right column

`e b`
calls the editor with the concatenation of left and right

`e` calls the editor with a zero length file

`q` exits from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

EXAMPLES

If `file1` contains:

```
x
a
b
c
d
```

and `file2` contains:

```
y
a
d
c
```

then the command:

```
sdiff file1 file2
```

would print:

```
x       |       y
a               a
b       <
c       <
d               d
        >       c
```

FILES

`/usr/bin/sdiff`
Executable file

SEE ALSO

`bdiff`(1), `diff`(1), `ed`(1), `sccsdiff`(1)

NAME

`sed` — edits a stream of data

SYNOPSIS

`sed` [`-n`] `-e` *command-line-script* [*file*]...

`sed` [`-n`] `-f` *scriptfile* [*file*]...

ARGUMENTS

`-e` *command-line-script*
: Causes the script to be taken directly from the command line (*command-line-script*). These options accumulate, so many scripts can be used in one invocation of the command. If there is just one `-e` option and no `-f`*scriptfile* options, the `-e` option may be omitted. Note that all shell metacharacters must be quoted when a command line script is supplied, so care must be taken when using the `-e` option.

`-f` *scriptfile*
: Causes the script to be taken from file *scriptfile.*

file Specifies the file or files that are edited, then sent to the standard output.

`-n` Suppresses the default output: output will only be generated if explicitly asked for by certain `sed` commands (`p`, `P`, `i`, `r`, and the `p` option of the `s` command).

DESCRIPTION

`sed` copies the named *file*s (standard input default) to the standard output, edited according to a script of `sed` commands.

A script consists of editing commands, one per line, of the following form:

[*address*[, *address*]] *function*

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.

Replace *address* with 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 as follows:

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 newline *embedded* in the pattern space.

A period (`.`) matches any character except the *terminal* newline 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 nonselected pattern spaces by use of the negation function (`!`) (see below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

Replace *text* with one or more lines, all but the last of which end with a backslash (`\`) to hide the newline. Backslashes in such 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.

When *rfile* or *wfile* are used in a command, they 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 Appends. Places *text* on the output before reading the next input line.

(2)`b` *label*
Branches to the `:` command bearing the *label.* If *label* is empty, branch to the end of the script.

(2)`c\`
text Changes. Deletes the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Starts the next cycle.

(2)`d`
Deletes the pattern space and starts the next cycle.

(2)D
: Deletes the initial segment of the pattern space through the first *newline*, then starts the next cycle.

(2)g
: Replaces the contents of the pattern space by the contents of the hold space.

(2)G
: Appends the contents of the hold space to the pattern space.

(2)h
: Replaces the contents of the hold space by the contents of the pattern space.

(2)H
: Appends the contents of the pattern space to the hold space.

(1)i\
text Inserts. Places *text* on the standard output.

(2)l
: Lists the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.

(2)n
: Copies the pattern space to the standard output. Replaces the pattern space with the next line of input.

(2)N
: Appends the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2)p
: Prints. Copies the pattern space to the standard output.

(2)P
: Copies the initial segment of the pattern space through the first newline to the standard output.

(1)q
: Quits. Branches to the end of the script. Does not start a new cycle.

(1)r *rfile*
: Reads the contents of *rfile*. Places them on the output before reading the next input line.

(2)s */regular expression/replacement/flags*
: Substitutes the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a more complete description, see ed(1). The *flags* option is

zero or more of:

n n= 1 - 512. Substitutes for just the nth occurrence of the *regular expression.*

g Substitutes globally. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Prints the pattern space if a replacement was made.

w *wfile*
Writes. Appends the pattern space to *wfile* if a replacement was made.

(2)t *label*
Tests. Branches 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*
Writes. Appends the pattern space to *wfile.*

(2)x
Exchanges the contents of the pattern and hold spaces.

(2)*y/string1/string2/*
Transforms. Replaces all occurrences of characters in *string1* with the corresponding character in *string2.* The lengths of *string1* and *string2* must be equal.

(2)! *function*
Applies the *function* (or group, if *function* is {) only to lines *not* selected by the address(es).

(0): *label*
Bears a *label* for b and t commands to branch to. This command does nothing.

(1)=
Places the current line number on the standard output as a line.

(2){
Executes the following commands through a matching } only when the pattern space is selected.

(0) Ignores an empty command.

(0)#
Treats the entire line as a comment with one exception, if this option appears as the first character on the first line of a script file. If the character after the # is an ``n'', then the default output will be

suppressed, as if the -n option had been invoked. The rest of the line after #n is also ignored. It is an error for the # command to be used on any line by the first line of the file. A script file must contain at least one noncomment line.

EXAMPLES

The following command will process `inputfile` according to the `sedfile` script, and place the results in `filea`:

```
sed -f sedfile inputfile > filea
```

The `sedfile` script:

```
4 a\
XXXXXXXXXXXXX
```

would insert a row of Xs after line 4.

WARNINGS

Operations based on a deleted line are lost. For example, if you insert text before line 4 and then delete line 4, the inserted text is lost. Reads at line 0 are actually reads before line 1, so deleting line 1 erases these reads. Writes are lost as well, although the filename is created.

FILES

`/bin/sed`
  Executable file

SEE ALSO

awk(1), ed(1), grep(1), lex(1)

NAME

`setfile` — sets attributes for Macintosh files, such as file type and creator

SYNOPSIS

`setfile` [`-a`*attribute-string*] [`-c`*creator*]
[`-l`*horizontal-pixels*`,` *vertical-pixels*] [`-t`*type*] [*data-file*]...

ARGUMENTS

`-a`*attribute-string*

Enables or disables any one or more Boolean attributes for a file through the inclusion of uppercase or lowercase letters in *attribute-string*. Use uppercase characters to enable the attribute and lowercase characters to disable the attribute:

`a` `A`	Enables or disables the switch-launch attribute (if possible).
`d` `D`	Enables or disables the desktop-located attribute.
`i` `I`	Enables or disables the initial attribute.
`l` `L`	Enables or disables the locked attribute.
`m` `M`	Enables or disables the shared attribute.
`s` `S`	Enables or disables the system attribute.
`v` `V`	Enables or disables the invisible attribute.

`-c`*creator*

Sets the creator attribute for the file to the four-letter string *creator*. Replace *creator* with the actual strings used by a Macintosh application. *creator* code includes spaces, tabs, or metacharacters. Enclose the code in quotation marks.

data-file

Specifies the Macintosh file to be changed. If you are working with a pair of AppleDouble files, specify the name of the data file only, not the name of the header file. The `setfile` command automatically looks for the associated header file, which should have the same name as the data file, but with a percent sign (`%`) prefix. If `setfile` cannot find the header file, it creates one.

`-l`*horizontal-pixels*, *vertical-pixels*
: Sets the horizontal and vertical coordinates for the icon location to *horizontal-pixels* pixels from the left extreme and *vertical-pixels* pixels from the upper extreme.

`-t`*type*
: Sets the type attribute for the file to the four-letter string. Replace *type* with the actual strings used by a Macintosh application. *type* code includes spaces, tabs, or metacharacters. Enclose the code in quotation marks.

DESCRIPTION

`setfile` sets the file type and creator of an AppleSingle file or the header file of an AppleDouble pair. See *Inside Macintosh,* Volume III, for a description of file types and creators.

Most Macintosh applications open a document file only if they recognize the type and creator. The Macintosh OS stores a file's type and creator in the directory. A/UX stores the type and creator as an entry in either an AppleSingle file or the header file of an AppleDouble pair. When an A/UX Toolbox application creates a file by using the normal File Manager routines, it automatically creates an AppleSingle file with the appropriate type and creator. (See the description of the File Manager in Chapter 4 of *Inside Macintosh.*)

The `setfile` command is useful when a file's type and creator are lost during a file transfer from the Macintosh environment to A/UX. The `setfile` program is also useful when you want to use a data file that was created by the standard A/UX file system instead of the A/UX Toolbox File Manager.

The usual symptom of an incorrect type is the file's failure to appear in the Open Standard File dialog box.

EXAMPLES

This command establishes the type of the data files 'report' and 'house' as `PNTG` and the creator as `MPNT`:

```
setfile -t PNTG -c MPNT report house
```

FILES

`/mac/bin/setfile`
: Executable file

SEE ALSO

`derez`(1), `rez`(1)

NAME

sh, rsh — runs the Bourne shell

SYNOPSIS

sh [-c *string*] [-i] [-r] [-s] [-a] [-e] [-f] [-h] [-k] [-n] [-t] [-u] [-v] [-x] [*args*]...]

rsh [-c *string*] [-i] [-r] [-s] [-a] [-e] [-f] [-h] [-k] [-n] [-t] [-u] [-v] [-x] [*args*]...]

ARGUMENTS

-a Marks variables which are modified or created for export.

-c *string*
Reads commands from the *string*.

-e Exits immediately if a command exits with a nonzero exit status.

-f Disables filename generation.

-h Locates and remembers function commands as functions are defined (function commands are normally located when the function is executed).

-i Causes the shell to be interactive if this option is present or if the shell input and output are attached to a terminal. 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.

-k Places all keyword arguments in the environment for a command, not just those that precede the command name.

-n Reads commands, but does not execute them.

-r Invokes a restricted shell.

-s Reads commands from the standard input if this option is present or if no arguments remain. Any remaining arguments specify the positional parameters. Shell output, except for special commands (see ''Special Commands''), is written to file descriptor 2.

-t Exits after reading and executing one command.

-u Treats unset variables as an error when substituting.

-v Prints shell input lines as they are read.

-x Prints commands and their arguments as they are executed.

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.

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 nonblank 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`(3) for a list of status values).

A `pipeline` is a sequence of one or more commands separated by ``|'' (or, for historical compatibility, by ^). The standard output of each command but the last is connected by a `pipe`(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by a `;`, `&`, `&&`, or `||`, and optionally terminated by a `;` 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 (nonzero) 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`
: Sets *name* to the next *word* taken from the `in` *word* list, each time a command is executed. 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`
: 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 ‘‘Filename 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`
: Executes the *list* following `if`. 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`
: 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*)
: Executes *list* in a subshell.

{*list*}
: Executes *list.*

name `()` {*list*}
: Defines 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 recognized only 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 newline to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (`‘ ’`) may be used as part or all of a word; trailing newlines are removed.

Parameter Substitution

The character `$` is used to introduce substitutable parameters. There are two types of parameters, positional and keyword. If `parameter` is a digit, it is a positional parameter. Positional parameters may be assigned values by `set`. Keyword parameters (also known as variables) may be assigned values by writing:

name=value [*name=value*]

Pattern-matching is not performed on `value`. There cannot be a function and a variable with the same `name`.

`${`*parameter*`}`
: The value, if any, of the parameter is substituted. The braces are required only when `parameter` is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If `parameter` is `*` or `@`, all the positional parameters, starting with `$1`, are substituted (separated by spaces). Parameter `$0` is set from argument zero when the shell is invoked.

`${`*parameter* `:-`*word*`}`
: If `parameter` is set and is non-null, substitute its value; otherwise substitute `word`.

`${`*parameter* `:=`*word*`}`
: If `parameter` is not set or is null, set it to `word`; the value of the parameter is substituted. Positional parameters may not be assigned 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 command, `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 checks only whether `parameter` is set or not.

The following parameters are set automatically 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). You may not change `PATH` if executing under `rsh`.

`CDPATH`
: The search path for the `cd` command.

`MAIL`
: If you have set this parameter to the name of a mail file and you have not set the `MAILPATH` parameter, the shell informs you 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 this parameter is set to 0, the shell will check before each prompt.

`MAILPATH`
: A colon-separated (`:`) list of filenames. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each filename may 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 newline.

`SHACCT`
: If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. Accounting routines such as `acctcom`(1) and `acctcms`(1M) can be used to analyze the data collected.

`SHELL`
: When the shell is invoked, it scans the environment (see

''Environment,'' below) for this name. If it is found and there is an r in the filename 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.

Filename Generation

Following substitution, each command word is scanned for the characters *, ?, and [. If one of these characters appears, the word is regarded as a pattern. The word is replaced with alphabetically-sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The character . at the start of a filename 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:

; & ( ) | ^ < > *newline space tab*

A character may be quoted (i.e., made to stand for itself) by preceding it with a \. The pair \newline is ignored. All characters enclosed between a pair of single quote marks (' '), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, ', , and $. The string, $*, is equivalent to $1 $2 ..., whereas $@ is equivalent to $1 2 ....

Prompting

When used interactively, the shell prompts with the value of PS1 before reading a command. If, at any time, a newline is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of PS2) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a command and are not passed on to the invoked command; substitution occurs before `word` or `digit` is used:

<*word*
: Uses file *word* as standard input (file descriptor 0).

>*word*
: Uses 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*
: Uses file *word* as standard output. If the file exists, output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

`<<[-]`
: *word* The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) \newline is ignored, and \\ must be used to quote the characters \\, $, ', and the first character of *word*. If - is appended to <<, all leading tabs are stripped from *word* and from the document.

<&*digit*
: Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using >&*digit*.

<&-
: The standard input is closed. Similarly for the standard output using >&-.

If any of the above is preceded by a *digit*, the file descriptor which will be associated with the file is that specified by the *digit* (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

```
...1>xxx2>&1
```

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

If a command is followed by `&`, the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell, as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment

The *environment* (see `environ`(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the values 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` option is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following command first prints `a=b c` and then, after the `-k` option is set, prints only `c`:

```
echo a=b c #first time prints a=b  c
set -k #puts all keyword args in env
echo a=b c #nowprintsonly" c; a=b goes to env
```

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 a command name matches one of the special commands listed below (see ''Special Commands''), 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 that this differs from the execution of shell procedures, which takes place in subshells). 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 pathname, which can appear immediately after the equals sign or between the colon delimiters anywhere else in 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 subshell is spawned to read it. A parenthesized command is also executed in a subshell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary `exec`s later). If the command was found in a relative directory, its location must be redetermined 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.

`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 (i.e., the empty string, specifying the current directory). Note that the current directory is specified by a null pathname, which can appear immediately after the equals 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`.

`continue` [*n*]
: Resume the next iteration of the enclosing `for` or `while` loop. If *n* is specified, resume at the *n*-th enclosing loop.

`echo` [*arg*...]
: Echo arguments. Arguments are written separated by blanks and terminated by a newline on the standard output. It understands C-like escape conventions.

`eval` [*arg*...]
: The arguments are read as input to the shell and the resulting command(s) executed.

`exec` [*arg*...]
: The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

`exit` [n]
: Causes a shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed (an end-of-file will also cause the shell to exit).

`export` [name...]
: The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. Function names may *not* be exported.

`hash [-r]`*[name...]*
: For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The

`-r` option causes the shell to forget all remembered locations. If no arguments are given, information (*hits* and *cost*) about remembered commands is presented. *hits* is the number of times a command has been invoked by the shell process. *cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (`*`) adjacent to the *hits* information. *cost* will be incremented when the recalculation is done.

`newgrp` *[arg...]*

Equivalent to `exec newgrp` *arg...* 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).

With no arguments, `newgrp` changes the group identification back to the group specified in the user's password file entry.

If the first argument to `newgrp` is a `-`, the environment is changed to what would be expected if the user actually logged in again.

This built-in version executes faster than the A/UX command `newgrp`(1) but is otherwise identical.

`pwd`

Print the current working directory. This built-in version executes faster than the A/UX command `pwd`(1) but is otherwise identical.

`read` [*name...*]

One line is read from the standard input and 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*. The return code is 0 unless an end-of-file is encountered.

`readonly`[*name...*]

The given *name*s are marked *readonly* and the values of the these *name*s 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*]...]

See the ''Arguments'' section at the beginning of this manual page for the argument descriptions to the `set` command. Using + rather than - causes these options to be turned off. These options can also be used upon invocation of the shell. The current set of options 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`

Evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a nonzero (false) exit status is returned; `test` also returns a nonzero exit status if there are no arguments. The superuser is always granted execute permission even though (1) execute permission is meaningful only for directories and regular files, and (2) `exec` requires that at least one execute mode bit be set for a regular file to be executable. The following primitives are used to construct *expr*:

`-r` *file*
Returns true if *file* exists and is readable.

`-w` *file*
Returns true if *file* exists and is writable.

`-x` *file*
Returns true if *file* exists and is executable.

`-f` *file*
Returns true if *file* exists and is a regular file.

`-d` *file*
Returns true if *file* exists and is a directory.

`-c` *file*
Returns true if *file* exists and is a character special file.

`-b` *file*
Returns true if *file* exists and is a block special file.

`-p` *file*
Returns true if *file* exists and is a named pipe (FIFO).

`-u` *file*
Returns true if *file* exists and its set user ID bit is set.

`-g` *file*
Returns true if *file* exists and its set group ID bit is set.

`-k` *file*
Returns true if *file* exists and its sticky bit is set.

`-s` *file*
Returns true if *file* exists and has a size greater than zero.

`-t` [*fildes*]
Returns true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.

`-z` *s1*
Returns true if the length of string *s1* is zero.

`-n` *s1*
Returns true if the length of the string *s1* is nonzero.

s1 `=` *s2*
Returns true if strings *s1* and *s2* are identical.

s1 `!=` *s2*
Returns true if strings *s1* and *s2* are *not* identical.

s1 Returns true if *s1* is *not* the null string.

n1 `-eq` *n2*
Returns true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons `-ne`, `-gt`, `-ge`, `-lt`, and `-le` may be used in place of `-eq`.

These primaries may be combined with the following operators:

`!` unary negation operator.

`-a` binary AND operator.

`-o` binary OR operator (`-a` has higher precedence than `-o`).

`(`*expr*`)`
parentheses for grouping.

Notice that all the operators and options are separate arguments to `test`. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

`test` is typically used in shell scripts as in the following example, which prints the message ''`foo is a directory`'' if it is found to be one when `test` is run. For example,

```
if test -d foo
then
        echo "foo is a dir"
fi
```

`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` [f][n]
: Imposes a size limit of *n*. The `-f` option imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed. If no options option is given, `-f` is assumed.

`umask`[*nnn*]
: The user file-creation mask is set to *nnn* (see `umask`(2)). If *nnn* is omitted, the current value of the mask is printed.

`unset`*[name*...]
: For each *name*, remove the corresponding variable or function. The variables `PATH`, `PS1`, `PS2`, `MAILCHECK`, and `IFS` cannot be unset.

`wait` [*n*]
: Waits for the specified process and reports its termination status. If *n* is not given, all currently active child processes are waited for and the return code is zero.

Invocation

If the shell is invoked through `exec`(2) and the first character of argument zero is `-`, commands are read initially 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 `-c`, `-i`, `-r`, and `-s` options are interpreted by the shell on invocation only. Note that, unless the `-c` or `-s` option 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.

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`
> `setting the value of` $PATH
> specifying path or command names containing **/**
> redirecting output (**>** and **>>**)

The restrictions above are enforced after `.profile` is interpreted.

When a command to be executed is found to be a shell procedure, `rsh` invokes `sh` to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the `.profile` has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., `/usr/rbin`) that can be safely invoked by `rsh`. Some systems also provide a restricted editor `red`.

EXAMPLES

Enter the command:

```
sh -x script1
```

to execute each command in `script1`, echoing the command just before executing it.

STATUS MESSAGES AND VALUES

Errors detected by the shell, such as syntax errors, cause the shell to return a nonzero exit status. If the shell is being used noninteractively, 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).

WARNINGS

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 pathname to correct this situation.

LIMITATIONS

Filename pattern matching is not done on redirected I/O filenames.

FILES

`/bin/sh`
    Executable file
`/etc/profile`
    Profile file
`$HOME/.profile`
    Profile file for the home directory
`/tmp/sh*`
    Temporary file
`/dev/null`
    Temporary file

SEE ALSO

`csh`(1), `chsh`(1), `echo`(1), `env`(1), `ksh`(1), `login`(1), `newgrp`(1), `pwd`(1)

`acctcms`(1M), `acctcom`(1M) in *A/UX System Administrator's Reference*

`dup`(2), `exec`(2), `fork`(2), `pipe`(2), `ulimit`(2), `umask`(2), `wait`(2), `signal`(3), `a.out`(4), `profile`(4), `environ`(5) in *A/UX Programmer's Reference*

''Bourne Shell Reference'' in *A/UX Shells and Shell Programming*

NAME

shl — manages the layering of multiple shells

SYNOPSIS

shl

DESCRIPTION

shl allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as layers, using the commands described below.

The current layer is the layer which can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal.

The stty character swtch (set to CONTROL-z if NULL) is used to switch control to shl from a layer. The shl program 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 and ioctl. Each layer has its own process group ID.

> *Note:* Only one instance of shell layering may be invoked in any given login session.

Definitions

A *name* is a sequence of characters delimited by a blank, tab or newline. 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*]
: Creates a layer called *name* and makes 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*]...
: Blocks the output of the corresponding layer when it is not the current layer, for each *name*.

delete *name* [*name*]...
: Deletes the corresponding layer for each *name*. All processes in the process group of the layer are sent the SIGHUP signal (see

signal(3)).

help(or?)
: Prints the syntax of the shl commands.

layers [-l][*name*]...
: Lists the layer name and its process group for each name. The -l option produces a ps(1)-like listing. If no arguments are given, information is presented for all existing layers.

resume [*name*]
: (followed by RETURN). Makes the layer referenced by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

toggle
: (followed by RETURN). Resumes the layer that was current before the last current layer.

unblock *name* [*name*]...
: Does not block the output of the corresponding layer when it is not the current layer, for each *name*.

quit
: Exits shl. All layers are sent the SIGHUP signal.

name
: (followed by RETURN). Makes the layer referenced by *name* the current layer.

FILES

/usr/bin/shl
: Executable file

/dev/sxt/???
: Virtual tty device files

$SHELL
: Variable file containing pathname of the shell to use (default is /bin/sh).

SEE ALSO

sh(1), stty(1),

ioctl(2), signal(3), sxt(7) in *A/UX Programmer's Reference*

A/UX Shells and Shell Programming

NAME

`size` — displays section sizes of common object files

SYNOPSIS

`size` [`-d`] [`-o`] [`-V`] [`-x`] *file*...

ARGUMENTS

`-d` Displays numbers in decimal.

`-o` Displays numbers in octal.

`-V` Supplies the version information on the `size` command.

`-x` Forces hexadecimal output in shell scripts.

DESCRIPTION

The `size` command produces section size information for each section in the common object files. The name of the section is shown followed by its size in bytes, physical address, and virtual address.

STATUS MESSAGES AND VALUES

`size:` ***name***`: cannot open`
name cannot be read

`size:` ***name***`: bad magic`
name is not an object file

FILES

`/bin/size`
Executable file

SEE ALSO

`as`(1), `cc`(1), `ld`(1)

`a.out`(4) in *A/UX Programmer's Reference*

NAME

`sleep` — suspends the system for a specified interval of time

SYNOPSIS

`sleep` *time*

ARGUMENTS

time

Specifies the number of seconds you wish the system to be suspended. This argument must be less than 65536 seconds.

DESCRIPTION

`sleep` suspends execution for a specified number of 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
```

EXAMPLES

The script:

```
label:
      command >> x
      command >> x
      date >> x
      sleep 10
      goto label
```

would execute the two commands and append the results to file `x`, then sleep for 10 seconds and repeat the process.

FILES

`/bin/sleep`

Executable file

SEE ALSO

`alarm`(2), `sleep`(3C) in *A/UX Programmer's Reference*

NAME

`sno` — runs the SNOBOL interpreter

SYNOPSIS

`sno` [*file*]...

ARGUMENTS

file Specifies the file to be interpreted by the `sno` command.

DESCRIPTION

`sno` is a SNOBOL compiler and interpreter (with slight differences). The `sno` command obtains input from the concatenation of the named *file*s and the standard input. All input through a statement containing the label `end` is considered program and is compiled. The rest is available to `syspit`.

The `sno` command differs from SNOBOL in the following ways:

There are no unanchored searches. To get the same effect:

*a**b*
Performs unanchored search for *b*.

*a*x*b=xc*
Specifies an unanchored assignment.

There is no back referencing.

x=abc
*a*x*x*
Specifies an unanchored search for `abc`.

Function declaration is done at compile time by the use of the (nonunique) label `define`. Execution of a function call begins at the statement following the `define`. Functions cannot be defined at run time, and the use of the name `define` is preempted. There is no provision for automatic variables other than parameters. Here are a few examples:

```
define f( )
define f(a, b, c)
```

All labels except `define` (even `end`) must have a nonempty statement.

Labels, functions, and variables must all have distinct names. In particular, the nonempty statement on `end` cannot merely name a label.

If `start` is a label in the program, program execution will start there. If not, execution begins with the first executable statement; `define` is not an executable statement.

There are no built-in functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators `/` and `*` must be set off by spaces.

The right side of assignments must be nonempty.

Either `'` or `"` may be used for literal quotes.

The pseudo-variable `sysppt` is not available.

FILES

`/usr/bin/sno`
Executable file

SEE ALSO

`awk(1)`

SNOBOL, a String Manipulation Language, by D. J. Farber, R. E. Griswold, and I. P. Polonsky, *JACM* 11 (1964), pp. 21-30

NAME

soelim — eliminates the source commands from nroff input

SYNOPSIS

soelim [*file*]...

ARGUMENTS

file Specifies the file to be read by the soelim command.

DESCRIPTION

soelim reads the specified files or the standard input and performs the textual inclusion implied by the nroff directives of the form

```
.so somefile
```

when they appear at the beginning of input lines. This is useful since programs such as tbl do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of a single minus (-) is taken to be a filename corresponding to the standard input.

Note that inclusion can be suppressed by using ' instead of ., such as

```
'so /usr/lib/tmac.s
```

EXAMPLES

Here is a sample usage of the soelim command:

```
soelim exum?.n | tbl | nroff -mm | col | lp
```

LIMITATIONS

The format of the source commands must be consistent; exactly one blank must precede and no blanks follow the filename.

FILES

/usr/ucb/soelim
Executable file

SEE ALSO

col(1), eqn(1), nroff(1), tbl(1), troff(1)

NAME

`sort` — sorts or merges files

SYNOPSIS

`sort` [`-b`] [`-c`] [`-d`] [`-f`] [`-i`] [`-m`] [`-M`] [`-n`] [`-o` *output*] [`-r`] [`-t`*x*] [`-u`] [`-y` [*kmem*]] [`-z`*recsz*] [+*pos1* [-*pos2*]] [*file*...]

ARGUMENTS

`-b` Ignores leading blanks when determining the beginning and ending positions of a restricted sort key. If the `-b` option is specified before the first +*pos1* argument, it is applied to all +*pos1* arguments. Otherwise, the `b` option may be attached independently to each +*pos1* or -*pos2* argument (as shown later).

`-c` Checks that the input file is sorted according to the ordering rules. This option gives no output unless the file is out of sort.

`-d` Uses ''dictionary'' order. Only letters, digits, and blanks (spaces and tabs) are significant in comparisons.

`-f` Folds lowercase letters into uppercase.

file Specifies the file containing the information to be sorted.

`-i` Ignores characters outside the ASCII range 040-0176 in non-numeric comparisons.

`-m` Merges only because the input files are already sorted.

`-M` Compares as months. The first three nonblank characters of the field are folded to uppercase and compared so that JAN < FEB < ... < DEC. Invalid fields compare low to JAN. The `-M` option implies the `-b` option (see later in this section).

`-n` Sorts by arithmetic value an initial numeric string, consisting of optional blanks, an optional minus sign, and zero or more digits with optional decimal point. The `-n` option implies the `-b` option (as described later). Note that the `-b` option is only effective when restricted sort-key specifications are in effect.

`-o` *output*
Places the output in the file *output* instead of in the standard output. This file may be the same as one of the inputs. There may be optional blanks between `-o` and *output*.

+*pos1*
-*pos2*
Restricts a sort key to one beginning at *pos1* and ending just before *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing -*pos2* designates the end of the line. Both of these options, *pos1* and *pos2*,

have the form *m.n* optionally followed by one or more of the options `b`, `d`, `f`, `i`, `n`, `r`, where *m* specifies the number of fields to skip from the beginning of the line and *n* specifies the number of characters to skip beyond. Thus, 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` option is in effect, *n* is counted from the first nonblank in the *m*+1st field; *+m.0b* refers to the first nonblank character in the *m*+1st field.

`-r` Reverses the sense of comparisons.

`-t`*x* Uses *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.

`-u` Suppresses all but one (unique) line in each set of lines having equal keys.

`-y` *kmem*
Sorts using a specified amount of kilobytes of memory, *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 the value, *kmem*, `sort` starts using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum is used. Thus, `-y0` is guaranteed to start with minimum memory. By convention, `-y` (with no argument) starts with maximum memory.

`-z`*recsz*
Records in the sort phase the size of the longest line read 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 is used. Lines longer than the buffer size cause `sort` to terminate abnormally. Supplying the actual number of bytes (or some larger value) in the longest line to be merged prevents abnormal termination.

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 filename 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.

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 (as described later), the specified ordering options override all global ordering options for that key.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a newline. 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 by using the `-b` or `-t`*x* options.

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` option is in effect, *n* is counted from the last leading blank in the *m*+1st field; -*m*.*1b* refers to the first nonblank 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

To sort the contents of *infile* with the second field as the sort key, use the command

```
sort +1 -2 infile
```

To 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, use the command

```
sort -r -o outfile +1.0 -1.2 infile1 infile2
```

To sort, in reverse order, the contents of *infile1* and *infile2* using the first nonblank character of the second field as the sort key, use the command

```
sort -r +1.0b -1.1b infile1 infile2
```

To print the password file (see `passwd`(4)) sorted by the numeric user ID (the third colon-separated field), use the command

```
sort -t: +2n -3 /etc/passwd
```

To print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the `-um`, with just one input file, makes the choice of a unique representative from a set of equal lines predictable), use the command

```
sort -um +2 -3 infile
```

STATUS MESSAGES AND VALUES

The `sort` program comments and exits with nonzero status for various trouble conditions (for example, when input lines are too long), and for disorder discovered under the `-c` option.

When a newline character is missing from the last line of an input file, `sort` appends one, prints a warning message, and continues.

FILES

`/bin/sort`
: Executable file

`/usr/tmp/stm???`
: Temporary file

SEE ALSO

`comm`(1), `join`(1), `rev`(1), `sortbib`(1), `tsort`(1), `uniq`(1)

NAME

`sortbib` — sorts bibliographic database

SYNOPSIS

`sortbib` [`-s`*keys*] *database...*

ARGUMENTS

database
: Specifies the database containing the files to be sorted. No more than 16 databases may be sorted together at one time.

keys
: Specifies the user-specified keys that will be used when sorting files.

`-s` Enables you to specify new values for the *keys* argument. For instance, `-sATD` will sort by author, title, and date, while `-sA+D` will sort by all authors, and date. Sort keys past the fourth are not meaningful.

DESCRIPTION

`sortbib` sorts files of records containing `refer` key-letters by user-specified keys. Records may be separated by blank lines, or by `.[` and `.]` delimiters, but the two styles may not be mixed together. This program reads through each *database* and pulls out key fields, which are sorted separately. The sorted key fields contain the file pointer, byte offset, and length of corresponding records. These records are delivered using disk seeks and reads, so `sortbib` may not be used in a pipeline to read standard input. Records longer than 4096 characters will be truncated.

By default, `sortbib` alphabetizes by the first `%A` and the `%D` fields, which contain the senior author and date.

The `sortbib` program sorts on the last word on the %A line, which is assumed to be the author's last name. A word in the final position, such as `jr.` or `ed.`, will be ignored if the name beforehand ends with a comma. Authors with two-word last names or unusual constructions can be sorted correctly by using the `nroff` convention `\0` in place of a blank. A `%Q` field is considered to be the same as `%A`, except sorting begins with the first, not the last, word. `sortbib` sorts on the last word of the `%D` line, usually the year. It also ignores leading articles (like `A` or `The`) when sorting by titles in the `%T` or `%J` fields; it will ignore articles of any modern European language.

If a sort-significant field is absent from a record, `sortbib` places that record before other records containing that field.

LIMITATIONS

Records with missing author fields should probably be sorted by title.

FILES

`/usr/ucb/sortbib`
: Executable file

SEE ALSO

addbib(1), indxbib(1), lookbib(1), refer(1), roffbib(1)

NAME

`spell`, `hashmake`, `spellin`, `hashcheck` — find spelling errors

SYNOPSIS

`spell` [`-v`] [`-b`] [`-x`] [`-l`] [`+local-file`] [*file*]...

`hashmake`

`spellin` *n*

`hashcheck` *spelling-list*

ARGUMENTS

`+local-file`
: Specifies 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 the spelling list included in `spell`) for each job. Under the *+local-file* option, words found in *local-file* are removed from the output of `spell`.

`-b` Checks the British spelling. In addition to preferring `centre`, `colour`, `programme`, `speciality`, `travelled`, and so on, this option insists upon `-ise` in words like `standardise`.

file Specifies the user-specified file that is to be searched for spelling errors.

`-l` Causes the `spell` command to follow the chains of all included files.

n Specifies the number of hash codes to be read by the `spellin` routine.

spelling-list
: Specifies the compressed spelling list that is read by the `hashcheck` routine.

`-v` Causes all words not literally in the spelling list to be printed. Plausible derivations from the words in the spelling list are indicated.

`-x` Prints every plausible stem with = for each word.

DESCRIPTION

`spell` collects words from each named *file* and locates them in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, or suffixes) from words in the spelling list are printed on the standard output. If no *file* is named, words are collected from the standard input.

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`.

The spell command ignores most troff(1), tbl(1), and eqn(1) constructions.

Three hashmake, spellin, and hashcheck 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 hash codes from the standard input and writes a compressed spelling list on the standard output. Information about the hash coding is printed on standard error. The compressed spelling list from the spellin output is in binary format and should be generally redirected into a file or a pipe.

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. The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, it 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 and listed in the section ''FILES.'' Copies of all output are accumulated in the history file. The stop list filters out misspellings (for example, thier=thy-y+ier) that would otherwise pass.

EXAMPLES

Entering:

```
spell filea fileb filec > mistakes
```

would put a list of the words from filea, fileb, and filec that were not part of the on-line dictionary into the file mistakes.

The following example creates the hashed spelling list hlist and checks the result by comparing the two temporary files; they should be equal.

```
cat wds | /usr/lib/spell/hashmake | sort -u >tmp1
cat tmp1 | /usr/lib/spell/spellin 'cat tmp1 | wc -l' >hlist
cat hlist | /usr/lib/spell/hashcheck >tmp2
diff tmp1 tmp2
```

LIMITATIONS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions. Typically, these are kept in a separate local file that is added to the hashed *spelling-list* via spellin.

The British spellings are incomplete.

FILES

`/bin/spell`
    Executable file
`/usr/lib/spell`
    Executable file
`/usr/lib/spell/spellin`
    Executable file
`/usr/lib/spell/hashcheck`
    Executable file
`/usr/lib/spell/hashmake`
    Executable file
`D_SPELL=/usr/lib/spell/hlist[ab]`
    Executable file
`S_SPELL=/usr/lib/spell/hstop`
    Executable file
`H_SPELL=/usr/lib/spell/spellhist`
    Spell history file
`/usr/lib/spell/spellprog`
    Executable file
`/usr/lib/spell/compress`
    Executable file

SEE ALSO

`diction`(1), `deroff`(1), `eqn`(1), `sed`(1), `sort`(1), `style`(1), `tbl`(1), `tee`(1), `troff`(1)

See spell(1)

NAME

spline — interpolates a smooth curve

SYNOPSIS

spline [-a] [-k] [-n] [-p] [-x*lower* [*upper*]]

ARGUMENTS

-a Supplies 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 Sets the constant *k* used in the boundary value computation:

$$y''_0 = ky''_1, \quad y''_n = ky''_{n-1}$$

by the next argument (default $k = 0$).

-n Spaces output points so that approximately *n* intervals occur between the lower and upper *x* limits (the default is $n = 100$).

-p Makes output periodic, i.e., match derivatives at ends. First and last input values should normally agree.

-x[*lower* [*upper*]]
Specifies the 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).

DESCRIPTION

spline 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 (R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed., pp. 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted.

EXAMPLES

The command:

```
spline -n 10 > spline.out
0 0
1 2
2 4
3 9
```

will create the file spline.out with the contents:

```
3.000000   8.999999
2.666667   7.096296
2.333333   5.370370
2.000000   4.000000
```

```
1.666667   3.096296
1.333333   2.503703
1.000000   2.000000
0.666667   1.407407
0.333333   0.725926
0.000000   0.000000
```

STATUS MESSAGES AND VALUES

When data is not strictly monotone in *x*, `spline` reproduces the input without interpolating extra points.

LIMITATIONS

A limit of 1,000 input points is enforced silently.

FILES

`/usr/bin/spline`
Executable file

SEE ALSO

`graph`(1G), `tplot`(1G)

NAME

split — splits a file into a specified number of pieces

SYNOPSIS

split [-] [-*n*] [*file* [*output-file*]...]

ARGUMENTS

- Causes split to read the standard input.

file Specifies the user-specified input file that is read by split. If no file is specified, split reads the standard input.

-*n* Specifies the number of line pieces the input file should be split into. The default is 1000 lines.

output-file
Specifies the name of the output file. The name of the first output file has aa appended, and so on lexicographically, up to zz (a maximum of 676 files). The output file cannot be longer than 12 characters. If no output filename is given, x is used, which is the default.

DESCRIPTION

split reads from the specified file and writes it in the specified number of line pieces onto a set of output files.

EXAMPLES

The command:

```
split -100 filea newfile
```

would split filea into 100-line pieces and put them in newfileaa, newfileab, and so forth until the end of filea.

FILES

/usr/bin/split
Executable file

SEE ALSO

bfs(1), csplit(1), fsplit(1)

NAME

`ssp` — produces single spaced output

SYNOPSIS

`ssp [-]` [*name*]...

ARGUMENTS

`-` Removes all blank lines.

name
: Specifies the name of the input file to be affected.

DESCRIPTION

`ssp` removes extra blank lines from its input, compressing two or more blank lines into one. Note that if a line contains any characters at all (including spaces or tabs), then `ssp` does not considered it to be blank. `ssp` can be used directly, or as a filter after `nroff` or other text formatting operations.

EXAMPLES

The command:

```
nroff -ms filea fileb | ssp > filec
```

would `nroff` the files with the `-ms` macro package, then single space the output and put it into `filec`.

FILES

`/usr/bin/ssp`
: Executable file

SEE ALSO

`awk`(1), `sed`(1)

NAME

`strings` — finds the printable strings in an object or other binary file

SYNOPSIS

`strings` [`-`] [`-o`] [`-`*number*] *file...*

ARGUMENTS

`-` Causes `strings` to only look in the initialized data space of object files, unless this option is given.

file Specifies the binary file to be searched.

`-o` Causes each string to be preceded by its offset in the file (in octal).

`-`*number*
Causes the specified number to be used as the minimum string length, rather than 4.

DESCRIPTION

`strings` looks for ASCII strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null.

The `strings` program is useful for identifying random object files and many other things.

EXAMPLES

Entering the command:

```
strings obj1
```

will locate the ASCII-character strings in the object file `obj1`.

LIMITATIONS

The algorithm for identifying strings is extremely primitive.

FILES

`/bin/strings`
Executable file

SEE ALSO

od(1), `xstr`(1)

NAME

`strip` — strips symbol and line number information from an object file

SYNOPSIS

`strip` [`-l`] [`-r`] [`-s`] [`-V`] [`-x`] *file...*

ARGUMENTS

file Specifies the object file to be stripped.

`-l` Strips line number information only; does not strip any symbol table information.

`-r` Resets the relocation indexes into the symbol table.

`-s` Resets the line number indexes into the symbol table (do not remove). Resets the relocation indexes into the symbol table.

`-V` Prints the version of the `strip` command executing on the standard error output.

`-x` Does not strip static or external symbol information.

DESCRIPTION

The `strip` command strips the symbol table and line number information from object files, including archives. When `strip` has been performed, no symbolic debugging access is available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using the options.

If there are any relocation entries in the object file and any symbol table information is to be stripped, `strip` complains and terminates without stripping *file* unless the `-r` option is used.

If the `strip` command is executed on a common archive file (see `ar`(4)) the archive symbol table is removed. The archive symbol table must be restored by executing the `ar`(1) command with the `s` option before the archive can be link edited by the `ld`(1) command. `strip` instructs the user with appropriate warning messages when this situation arises.

The purpose of this command is to reduce the file storage overhead taken by the object file.

STATUS MESSAGES AND VALUES

`strip:` *name*`: cannot open`
name cannot be read.

`strip:` *name*`: bad magic`
name is not an object file.

`strip:` *name*`: relocation entries present`
: *name* contains relocation entries and the `-r` option was not used; therefore, the symbol table information cannot be stripped.

FILES

`/bin/strip`
: Executable file

`/usr/tmp/str??????`
: Temporary file

SEE ALSO

`as`(1), `cc`(1), `ld`(1), `sdb`(1)

`ar`(4), `a.out`(4) in *A/UX Programmer's Reference*

NAME

`stty` — sets the modes for a terminal

SYNOPSIS

`stty` [`-a`] [`-g`] [`-n` *file*] [*options*]

ARGUMENTS

`-a` Causes `stty` to report all of the option settings.

`-g` Reports current settings in a form that can be used as an argument to another `stty` command.

`-n` *file*
Causes `stty` to open the file specified by *file* with the option `O_NODELAY` and uses it as standard input. (This means that it will open modem-controlled lines immediately instead of waiting for a carrier.)

options
Specifies the various options listed in one of the five groups under the ``DESCRIPTION'' section.

DESCRIPTION

`stty` sets certain terminal I/O modes for the device that is the current standard input; without arguments, it reports the settings of certain modes.

Detailed information about the modes listed in the groups ``Control Modes'', ``Input Modes'', ``Output Modes'', ``Local Modes'', and ``Control Assignments'' may be found in `termio`(7). Options in the ``Combination Modes'' group are implemented by using *options* in any of those five groups. Note that many combinations of *options* make no sense; however, no checking is performed.

Control Modes

`parenb` (`-parenb`)
Enables (disables) parity generation and detection.

`parodd` (`-parodd`)
Selects odd (even) parity.

`cs5 cs6 cs7 cs8`
Selects character size; see `termio`(7).

`0` Hangs up the phone line immediately.

`50 75 110 134 150 200 300 600 1200`
`1800 2400 4800 9600 exta extb`
Sets terminal baud to the number given, if possible. (All speeds are not supported by all hardware interfaces; 9600 baud is assumed.) `19200` is equivalent to `exta`. `38400` is equivalent to `extb`.

`hupcl` (`-hupcl`)
: Hangs up (does not hang up) modem connection on last close.

`hup` (`-hup`)
: Works the same as `hupcl` (`-hupcl`).

`cstopb` (`-cstopb`)
: Uses two (one) stop bits per character.

`cread` (`-cread`)
: Enables (disables) the receiver.

`clocal` (`-clocal`)
: Assumes a line without (with) modem control.

`loblk` (`-loblk`)
: Blocks (does not block) output from a noncurrent layer.

Input Modes

`ignbrk` (`-ignbrk`)
: Ignores (does not ignore) break on input.

`brkint` (`-brkint`)
: Signals (does not signal) `INTR` on break.

`ignpar` (`-ignpar`)
: Ignores (does not ignore) parity errors.

`parmrk` (`-parmrk`)
: Marks (does not mark) parity errors; see `termio`(7).

`inpck` (`-inpck`)
: Enables (disables) input parity checking.

`istrip` (`-istrip`)
: Strips (does not strip) input characters to seven bits.

`inlcr` (`-inlcr`)
: Maps (does not map) NL to CR on input.

`igncr` (`-igncr`)
: Ignores (does not ignore) CR on input.

`icrnl` (`-icrnl`)
: Maps (does not map) CR to NL on input.

`iuclc` (`-iuclc`)
: Maps (does not map) uppercase alphabetics to lowercase on input.

`ixon` (`-ixon`)
: Enables (disables) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

`ixany` (`-ixany`)
: Allows any character (only DC1) to restart output.

`ixoff` (`-ixoff`)
: Requests that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

`opost` (`-opost`)
: Post-processes output (does not post-process output; ignores all other output modes).

`olcuc` (`-olcuc`)
: Maps (does not map) lowercase alphabetics to uppercase on output.

`onlcr` (`-onlcr`)
: Maps (does not map) NL to CR-NL on output.

`ocrnl` (`-ocrnl`)
: Maps (does not map) CR to NL on output.

`onocr` (`-onocr`)
: Does not (does) output a CR at column zero.

`onlret` (`-onlret`)
: NL performs (does not perform) the CR function on the terminal.

`ofill` (`-ofill`)
: Uses fill characters (uses timing) for delays.

`ofdel` (`-ofdel`)
: Fills characters are DELs (NULs).

`cr0 cr1 cr2 cr3`
: Selects style of delay for returns; see `termio`(7).

`nl0 nl1`
: Selects style of delay for linefeeds; see `termio`(7).

`tab0 tab1 tab2 tab3`
: Selects style of delay for horizontal tabs; see `termio`(7).

`bs0 bs1`
: Selects style of delay for backspaces; see `termio`(7).

`ff0 ff1`
: Selects style of delay for form-feeds; see `termio`(7).

`vt0 vt1`
: Selects style of delay for vertical tabs; see `termio`(7).

Local Modes

`isig (-isig)`
: Enables (disables) the checking of characters against the special control characters INTR, QUIT, and SWTCH.

`icanon (-icanon)`
: Enables (disables) canonical input (ERASE and KILL processing).

`xcase (-xcase)`
: Canonicals (unprocessed) upper/lower-case presentation.

`echo (-echo)`
: Echoes back (does not echo back) every character typed.

`echoe (-echoe)`
: Echoes (does 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)`
: Echoes (does not echo) NL after KILL character.

`lfkc (-lfkc)`
: Works the same as `echok (-echok)`; obsolete.

`echonl (-echonl)`
: Echoes (does not echo) NL.

`noflsh (-noflsh)`
: Disables (enables) flush after INTR, QUIT, or SWTCH.

`stwrap (-stwrap)`
: Disables (enables) truncation of lines longer than 79 characters on a synchronous line.

`stflush (-stflush)`
: Enables (disables) flush on a synchronous line after every `write`(3).

`stappl (-stappl)`
: Uses application mode (uses line mode) on a synchronous line.

Control Assignments

`erase` *c*
: Sets the erase character to *c* (by default, sets to DELETE in the A/UX standard distribution).

`kill` *c*
: Sets the kill character to *c* (by default, sets to CONTROL-U in the

A/UX standard distribution).

`intr` *c*
: Sets the interrupt character to *c* (by default, sets to CONTROL-C in the A/UX standard distribution).

`quit` *c*
: Sets the quit character to *c* (by default, sets to CONTROL-| in the A/UX standard distribution).

`swtch` *c*
: Sets the switch character to *c* (by default, sets to CONTROL-` in the A/UX standard distribution).

`eof` *c*
: Sets the EOF character to *c* (by default, sets to CONTROL-D in the A/UX standard distribution).

`eol` *c*
: Sets the EOL character to *c* (by default, sets to CONTROL-` in the A/UX standard distribution).

`min` *c*
: Sets the min character to *c* (`min` is used only with `-icanon`; see `termio`(7).

`time` *c*
: Sets the time character to *c* (`time` is used only with `-icanon`; see `termio`(7).

  If *c* is preceded by a circumflex (`^`) appropriately escaped from the shell, then the value used is the corresponding control character (for example, `^d` is a CONTROL-D, `^?` is interpreted as DELETE, and `^-` is interpreted as undefined).

`line` *i*
: Sets the line discipline to *i* ($0 \leq i < 127$).

BSD 4.2 Compatible Features

`susp` *c*
: Sets the suspend character to *c*. When typed, the suspend character sends `SIGTSTP` to the current process group.

`dsusp` *c*
: Sends `SIGTSTP` to the current process group, when this option is set and a program attempts to read terminal input,

`tostop (-tostop)`
: Stops background processes which write on the control tty until brought into foreground by the shell, when this option is set.

Combination Modes

`evenp` or `parity`
: Enables `parenb` and `cs7`.

`oddp`
: Enables `parenb`, `cs7`, and `parodd`.

`-parity`, `-evenp`, or `-oddp`
: Disables `parenb`, and sets `cs8`.

`raw` (`-raw` or `cooked`)
: Enables (disables) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).

`nl` (`-nl`)
: Unsets (sets) `icrnl` and `onlcr`. In addition `-nl` unsets `inlcr`, `igncr`, `ocrnl`, and `onlret`.

`lcase` (`-lcase`)
: Sets (unsets) `xcase`, `iuclc`, and `olcuc`.

`LCASE` (`-LCASE`)
: Works the same as `lcase` (`-lcase`).

`tabs` (`-tabs` or `tab3`)
: Preserves (expands to spaces) tabs when printing.

`ek` Resets the ERASE and KILL characters back to normal DELETE and CONTROL-U.

`sane`
: Resets all modes to some reasonable values.

term
: Sets all modes suitable for the terminal type *term*, where *term* is one of `tty33`, `tty37`, `vt05`, `tn300`, `ti700`, or `tek`.

Hardware-specific Modes

`modem (-modem)`
: Enables (disables) modem control for this device. Normally, this is only turned on for lines connected to modems. Such lines cannot be opened; see `open`(2) unless the device's data carrier detect line (DCD) is asserted by an external device such as a modem. Not all devices support this option; refer to the specific device's documentation for details. This option is on by default for `/dev/modem` and `/dev/tty0`. Since it uses the same signal line as `dtrflow` and `emodem`, these options cannot be used at the same time.

`emodem (-emodem)`
: Enables (disables) ''European style'' modem control. Similar to

modem, as described previously. Refer to termio(7) for further information.

dtrflow (-dtrflow)
hxctl (-hxctl)
Enables hardware flow control for this device using the DCD line as input. This is normally used as a flow control with devices such as printers. Not all devices support this option; refer to the specific device's documentation for details. These options are on by default for /dev/printer and /dev/tty1. Since they use the same signal line as modem and emodem, dtrflow cannot be used at the same time as those options. Note that dtrflow and hxctl are synonymous and cannot be used at the same time.

flow (-flow)
Enables hardware flow control using the request to send and clear to send lines (RTS/CTS) on a serial device. Not all devices support this option; refer to the specific device's documentation for details. Often it is preferable and easier to use XON/XOFF (ixon, ixoff and ixany) which is supported for all devices.

The hardware-specific modes all apply to modem control; not all devices support all or any of them. If any of them are supported, then UIOCTTSTAT is supported. The default mode is UIOCNOMODEM/UIOCNOFLOW. All these are ''remembered'' when a device is closed and reopened again.

FILES

/bin/stty
Executable file

SEE ALSO

tabs(1)

ioctl(2), termio(7) in *A/UX Programmer's Reference*

NAME

`style` — analyzes the surface characteristics of documents

SYNOPSIS

`style` [`-a`] [`-e`] [`-l` *num*] [`-ml`] [`-mm`] [`-p`] [`-P`] [`-r` *num*] *file*...

ARGUMENTS

`-a` Prints all sentences with their length and readability index.

`-e` Prints all sentences that begin with an expletive.

file Specifies the file to be analyzed.

`-l` *num*
: Prints all sentences longer than the specified number, *num*.

`-ml`
: Causes `deroff` to skip lists. This option should be used if the document contains many lists of nonsentences.

`-mm`
: Overrides the default macro package `-ms` with the `-mm` macro package.

`-p` Prints all sentences that contain a passive verb.

`-P` Prints parts of speech of the words in the document.

`-r` *num*
: Prints all sentences whose readability index is greater than the specified number, *num*.

DESCRIPTION

`style` analyzes the surface characteristics of the writing style of a document. It reports on readability, sentence length and structure, word length and usage, verb type, and sentence openers. Because `style` runs `deroff` before looking at the text, formatting header files should be included as part of the input.

LIMITATIONS

Use of nonstandard formatting macros may cause incorrect sentence breaks.

FILES

`/usr/ucb/style`
: Executable file

`/usr/lib/style1`
: File containing style information

`/usr/lib/style2`
: File containing style information

`/usr/lib/style3`
: File containing style information

SEE ALSO

deroff(1), diction(1), spell(1)

NAME

su — substitutes user ID

SYNOPSIS

su [-] [*name*[*arg* ...]]

ARGUMENTS

-	Changes your environment to that of *name* for the temporary login session.
arg	Specifies the information that is 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 argument of -r will give the user a restricted shell.
name	Specifies the name of the user you wish to assume.

DESCRIPTION

su allows a user to become another user without logging off. The default user *name* is root (that is, superuser).

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 (CONTROL-D) to the new 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 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:/usr/etc 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(1), respectively. If the user's program is other than /bin/sh, then the program is invoked with an *arg0* of *-program* by both login(1) and su(1).

All attempts to become another user using su are logged in the log file /usr/adm/sulog.

EXAMPLES

The command

```
su joshua
```

would cause the system to prompt for `joshua`'s password; if the password is typed in correctly, `joshua`'s identity is substituted.

To become user `justin` while retaining the previously exported environment, execute

```
su justin
```

To become user `justin` but change the environment to what would be expected if `justin` had originally logged in, execute

```
su - justin
```

To execute *command* with the temporary environment and permission of user `justin`, type

su - justin -c "*command args*"

FILES

`/bin/su`
: Executable file

`/etc/passwd`
: Password file

`/etc/profile`
: Profile file

`$HOME.profile`
: Home directory profile file

`/usr/adm/sulog`
: Log of substitute users

SEE ALSO

csh(1), env(1), ksh(1), login(1), sh(1)

passwd(4), profile(4), environ(5) in Πm

NAME

subj — generates a list of subjects from documents

SYNOPSIS

subj *file...*

ARGUMENTS

file Specifies the file that is to be searched for subjects.

DESCRIPTION

subj searches *file*s for subjects that might be appropriate in a subject-page index and prints the list of subjects on the standard output. The document should contain formatting commands (from nroff, troff, and mm among others) to make the best use of subj.

WARNINGS

subj selects sequences of capitalized words as subjects except the first word in each sentence. Thus, if a sentence begins with a proper noun, the capitalization rule will not select this word as a subject. On the other hand, since each sentence is expected to begin on a newline, the first word of a sentence that begins in the middle of a line may be erroneously selected.

The output of subj may not be appropriate for your needs and should be edited accordingly.

LIMITATIONS

The subj program also selects as subjects modifier-noun sequences from the abstract, headings, and topic sentences (the first sentence in each paragraph), and occasionally a word is incorrectly categorized as a noun or adjective.

FILES

/usr/bin/subj
    Executable file

SEE ALSO

mm(1), ndx(1), troff(1)

NAME

`sum` — calculates a checksum

SYNOPSIS

`sum [-r]` *file...*

ARGUMENTS

file Specifies the file to be calculated by the `sum` command.

`-r` Causes an alternate algorithm to be used in computing the checksum.

DESCRIPTION

`sum` calculates and displays a 16-bit checksum for the named file, and also displays 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.

EXAMPLES

The command:

```
sum filea
```

produces the checksum and the block count of `filea`.

STATUS MESSAGES AND VALUES

`Read error` is indistinguishable from end-of-file on most devices; check the block count.

FILES

`/bin/sum`
    Executable file

SEE ALSO

`sumdir`(1), `wc`(1)

NAME

`sumdir` — sums and counts the characters within the files of the given directories

SYNOPSIS

`sumdir` [*directories*]

ARGUMENTS

directories

Specifies the directories that contain the files to be checked.

DESCRIPTION

`sumdir` provides a recursive checksum of all files in the specified directory. It calculates and prints a 16-bit checksum for the named file, and also prints the number of characters in the file.

It is typically used to look for bad spots on the file system, or to validate a file transmitted over some transmission line.

The output from this program differs from the output from the `sum`(1) program in that `sumdir` prints the number of characters rather than the number of blocks in the file.

EXAMPLES

The command:

```
sumdir man1
```

produces the checksum and the character count of the files in the `man1` directory.

FILES

`/usr/bin/sumdir`

Executable file

SEE ALSO

`sum`(1)

NAME

`sync` — updates the superblock

SYNOPSIS

`sync`

DESCRIPTION

`sync` executes the `sync` system primitive. If the system is to be stopped, `sync` must be called to insure file system integrity. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See `sync`(2) for details.

EXAMPLES

The command:

```
sync
```

should be typed to flush all internal disk buffers, before bringing down the system.

FILES

`/bin/sync`
Executable file

SEE ALSO

`shutdown`(1M) in *A/UX System Administrator's Reference*

`sync`(2) in *A/UX Programmer's Reference*

NAME

`sysline` — displays the system status on the status line of a terminal

SYNOPSIS

`sysline` [+*seconds*] [`-b`] [`-c`] [`-d`] [`-D`] [`-e`] [`-h`] [`-H` *remote*] [`-i`] [`-j`] [`-l`] [`-m`] [`-p`] [`-q`] [`-r`] [`-s`]

ARGUMENTS

+*seconds*
: Specifies the interval (in seconds) in which the status line is updated. The default is 60 seconds.

`-b` Beeps once every half hour and twice every hour, just like those obnoxious watches you keep hearing.

`-c` Clears the status line for 5 seconds before each redisplay.

`-d` Debugs and prints status line data in human-readable format.

`-D` Prints out the current day/date before the time.

`-e` Prints out only the information. Does not print out the control commands necessary to put the information on the bottom line. This option is useful for putting the output of `sysline` onto the mode line of an `emacs` window.

`-h` Prints out the host machine's name after the time.

`-H` *remote*
: Prints the load average on the remote host *remote*. If the host is down, or is not sending out `rwhod` packets, then the down time is printed instead.

`-i` Prints out the process ID of the `sysline` process onto standard output upon startup. With this information you can send the alarm signal to the `sysline` process to cause it to update immediately. `sysline` writes to the standard error, so you can redirect the standard output into a file to catch the process id.

`-j` Forces the sysline output to be left-justified even on terminals capable of cursor movement on the status line.

`-l` Does not print the names of people who log in and out.

`-m` Does not check for mail.

`-p` Does not report the number of processes, which are runnable and suspended.

`-q` Does not print out diagnostic messages if something goes wrong when starting up.

`-r` Does not display in reverse video.

`-s` Prints ''short'' form of line by left-justifying. *iff* escapes are not allowed in the status line. Some terminals (the `tvi`'s and `Freedom 100`'s, for example) do not allow cursor movement (or other ''intelligent'' operations) in the status line. For these terminals, `sysline` normally uses blanks to cause right-justification. This option will disable the adding of the blanks.

DESCRIPTION

`sysline` runs in the background and periodically displays system status information on the status line of the terminal. Not all terminals have a status line. Those that do include the `h19`, `c108`, `aaa`, `vt100`, `tvi925/tvi950`, and `Freedom 100`. Of these, only the `h19` termcap entry supports the status line.

Note: The Macintosh monitor does not have a status line.

If no options are given, `sysline` displays the time of day, the current load average, the change in load average in the last 5 minutes, the number of users (followed by `u`), the number of runnable processes (followed by `r`), the number of suspended processes (followed by `s`), and the users who have logged on and off since the last status report. Finally, if new mail has arrived, a summary of it is printed. If there is unread mail in your mailbox, an asterisk will appear after the display of the number of users. The display is normally in reverse video (if your terminal supports this in the status line) and is right-justified to reduce distraction. Every fifth display is done in normal video to give the screen a chance to rest.

If you have a file named `.who` in your home directory, then the contents of that file are printed first. One common use of this feature is to alias chdir, pushd, and popd to place the current directory stack in `~/.who` after it changes the new directory.

If you have a file `.syslinelock` in your home directory, then `sysline` will not update its statistics and write on your screen; it will just go to sleep for a minute. This is useful if you want to disable `sysline` momentarily. Note that it may take a few seconds from the time the lock file is created until you are guaranteed that `sysline` will not write on the screen.

LIMITATIONS

If you interrupt the display, you may find your cursor missing or stuck on the status line. The best thing to do is to reset the terminal.

If there is too much for one line, the excess is thrown away.

FILES

`/usr/ucb/sysline`
: Executable file

`/etc/termcap`
: Terminal capabilities file

`/etc/utmp`
: File containing names of people who are logged in

`/dev/kmem`
: File containing process table

`/usr/spool/rwho/whod.*`
: File containing `who/uptime` information for remote hosts

`${HOME}/.who`
: File containing information to print on bottom line

`${HOME}/.syslinelock`
: Lock file; when it exists, `sysline` will not print.

SEE ALSO

`ps`(1)

`pstat`(1M) in *A/UX System Administrator's Reference*

NAME

`systemfolder`, `systemfolder24` — create a personal System Folder

SYNOPSIS

`systemfolder` [`-f`] [`-u` *user*]

`systemfolder24` [`-f`] [`-u` *user*]

ARGUMENTS

`-f` Forces the update of the System file. By default, the System file is not updated if it already exists in the personal System Folder. This convention prevents overwriting of personal fonts, desk accessories, and so on.

`-u` *user*
Creates a personal System Folder for the user designated by *user*. This option is generally used only by the system administrator. In this case, the System Folder is created in the home directory of the user designated by *user*.

DESCRIPTION

`systemfolder` allows you to create a personal Macintosh System Folder in your home directory. When you enter the Finder environment, this personal System Folder is used instead of the global System Folder, `/mac/sys/System Folder`. Files are copied (privately modifiable) and linked (shared) from read-only versions stored in `/mac/lib/SystemFiles`. If a personal System Folder already exists, it is updated with any files from `SystemFiles` that are not present in the personal folder.

The `systemfolder24` command creates a personal System Folder specifically for use with 24-bit mode.

You can install a 24-bit System Folder to allow the use of certain desk accessories or inits that are not 32-bit clean. Note that you do not need a 24-bit System Folder to run 24-bit applications under the Finder (24-bit).

FILES

`$HOME/System Folder`
Directory corresponding to a new System Folder

`$HOME/System Folder24`
Directory corresponding to a new System Folder for 24 bit mode

`/mac/lib/SystemFiles/private/*`
Files that are copied into a new System Folder

`/mac/lib/SystemFiles/shared/*`
Files that become links in a new System Folder

See systemfolder(1)

SEE ALSO

startmac(1M) in *A/UX System Administrator's Reference*

NAME

`tabs` — sets the tab stops on a terminal

SYNOPSIS

`tabs` [*tabspec*] [+m[*n*]] [-T*type*]

ARGUMENTS

m[*n*]

Specifies the margin argument that is used for some terminals. This option 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 (left-most) margin on most terminals is obtained by `+m0`. The margin for most terminals is reset only when the `+m` option is given explicitly.

tabspec

Specifies the table specification. If no *tabspec* is given, the default value is `-8`, i.e., UNIX ''standard'' tabs. The lowest column number is 1. Note that for `tabs`, column 1 always refers to the left-most column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450. Replace *tabspec* with one of the following:

`-a` 1,10,16,36,72
Assembler, IBM S/370, first format

`-a2`
1,10,16,40,72
Assembler, IBM S/370, second format

`-c` 1,8,12,16,20,55
COBOL, normal format

`-c2`
1,6,10,14,49
COBOL, compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:

```
<:t-c2 m6 s66 d:>"
```

(see the `--`*file* option).

`-c3`
1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than `-c2`. This is the recommended format for COBOL. The

appropriate format specification is:

```
<:t-c3 m6 s66 d:>
```

(see the `--`*file* option).

`-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. Note that such a setting leaves a left margin of *n* columns on `TermiNet` terminals only . Of particular importance is the value `-8`: this represents the UNIX ''standard'' tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the `nroff -h` option for high-speed output. Another special case is the value `-0`, implying no tabs.

n1,n2,...
The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists `1,10,20,30` and `1,10,+10,+10` are considered identical.

`--`*file*
If the name of a file is given, `tabs` reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as `-8`. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the `pr`(1) command:

```
tabs --file; pr file
```

`-T`*type*
Specifies the terminal 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` option is supplied, `tabs` searches for the `$TERM` value in the *environment* (see `environ`(5)). If no *type* can be found, `tabs`

tries a sequence that will work for many terminals.

DESCRIPTION

`tabs` sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user must have remotely-settable hardware tabs.

Users of `TermiNet` terminals should be aware that they behave differently from most other terminals for some tab settings. The first number in a list of tab settings becomes the *left margin* on a `TermiNet` terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a `TermiNet` terminal, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

If the `-T`*type* or `+m`[*n*] option occurs more than once, the last value given takes effect.

Tab and margin setting is performed via the standard output.

EXAMPLES

This command will send commands to the terminal to set the tabs for COBOL format remotely:

```
tabs -c
```

The following command will set tabs in columns 6, 12 and 18:

```
tabs 6,12,18
```

The command:

```
tabs -10
```

will set tabs in columns 11, 21, 31, 41, 51, 61, and 71.

LIMITATIONS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

It is generally impossible to change the left margin usefully without also setting tabs.

The `tabs` command clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

STATUS MESSAGES AND VALUES

`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 predefined code cannot be found, where predefined codes include:

```
-a-a2-c-c2-c3-f-p-s-u
```

`can't open`
If the `--`*file* option is used, and file can't be opened.

`file indirection`
If the `--`*file* option is used and the specification in that file points to yet another file. Indirection of this form is not permitted.

FILES

`/usr/bin/tabs`
Executable file

SEE ALSO

`nroff`(1), `pr`(1), `tset`(1)

`term`(4), `environ`(5) in *A/UX Programmer's Reference*

NAME

tail — displays the last part of a file

SYNOPSIS

tail [±[*number*][[b][f]]] [*file*]

tail [±[*number*][[c][f]]] [*file*]

tail [±[*number*][[l][f]]] [*file*]

ARGUMENTS

±*number*
: Starts copying at the specified distance +*number* from the beginning, or the specified distance -*number* from the end of the input. If *number* is null, the value 10 is assumed. The specified distance is counted by lines (-l) by default.

b
: Counts the specified distance, ±*number*, in units of blocks.

c
: Counts the specified distance, ±*number*, in units of characters.

-f
: Enters an endless loop (wherein it sleeps for a second and then attempts to read and copy further records from the input file) instead of terminating after the line of the input file has been copied, if the input file is not a pipe. Thus it may be used to monitor the growth of a file that is being written by some other process. You must interrupt tail to escape this loop.

file
: Specifies the file to be used in the tail command. If no file is specified, the standard input is used.

l
: Counts the specified distance, ±*number*, in units of lines. This is the default.

DESCRIPTION

tail copies the named file to the standard output beginning at a designated place.

EXAMPLES

To print the last ten lines of the file jack, followed by any lines that are appended to jack between the time tail is initiated and interrupted, enter:

```
tail -f jack
```

The command:

```
tail -15cf jack
```

will print the last 15 characters of the file jack, followed by any lines that are appended to jack between the time tail is initiated and interrupted.

LIMITATIONS

Any `tails` relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

FILES

`/bin/tail`
: Executable file

SEE ALSO

`cat`(1), `dd`(1), `head`(1), `more`(1), `pg`(1)

NAME

talk — talks to another user via the terminal

SYNOPSIS

talk *user* [*ttyname*]

ARGUMENTS

user

Specifies the login name of the person you wish to talk to.

ttyname

Specifies the terminal that the person you wish to talk to is using.

DESCRIPTION

talk is a visual communication program that copies lines from your terminal to that of another user.

If you wish to talk to someone on your own machine, then *user* is just the login name of the person you wish to talk to. If you wish to talk to a user on another host connected via Ethernet to a local network running the TCP/IP networking software, *user* is of one of these forms:

host!*user*
host.*user*
host:*user*
user@*host*

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

While in a talk session, typing CONTROL-L will cause the screen to be reprinted, while your erase and kill characters will work in talk as normal.

To exit a talk session, just type your interrupt character; talk then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the mesg command. At the invocation of talk, talking is allowed. Certain commands, in particular nroff(1) and pr(1), disallow messages in order to prevent messy output.

EXAMPLES

When talk is first called, it sends the message

```
Message from TalkDaemon@his machine
talk: connection requested by your name@your machine.
talk: respond with: talk your name@your machine
```

to the user you wish to talk to. At this point, the recipient of the message should reply by typing `talk` *yourname@your machine*

It doesn't matter from which machine the recipient replies, as long as his login name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows.

FILES

`/usr/bin/talk`
Executable file

`/etc/hosts`
File containing list of host machines

`/etc/utmp`
File containing terminal information

SEE ALSO

`mail`(1), `mesg`(1), `who`(1), `write`(1)

NAME

`tar` — copies files to or from a `tar` archive

SYNOPSIS

`tar` [`-`]`c`[`0`...`7`[*density*]][`ilvbBdfs`] [*bBdfs-arg*]... *file*...

`tar` [`-`]`r`[`0`...`7`[*density*]][`ilvbBdfs`] [*bBdfs-arg*]... *file*...

`tar` [`-`]`t`[`0`...`7`][`ivw`][`f` *archive*] [*file-in-archive*]...

`tar` [`-`]`u`[`0`...`7`[*density*]][`ilvbBdfs`] [*bBdfs-arg*]... *file*...

`tar` [`-`]`x`[`0`...`7`][`timovw`[`f` *archive*] [*file-in-archive*]...

ARGUMENTS

`0`...`7`[*density*]

Selects a tape drive, where a digit between `0` and `7` is the drive number corresponding to a 9-track tape drive and *density* is the density to be established:

`-l` low (800 bpi)

`-m` medium (1600 bpi)

`-h` high (6250 bpi)

The default is `0m` (drive number 0 at 1600 bpi).

Note that the drive number is unrelated to SCSI ID numbers. The command option that can be used to select a particular SCSI device is `-f` (described later in this list).

`-b`

(`-c`, `-r`, `-u` modes)
Lets you specify the blocking factor for tape records as one of the supplied *bBdfs-arg* values. The default value is 1, the maximum is 20. This command option should only be used with raw magnetic tape devices (see the `-f` option). The block size is determined automatically when reading tapes.

`-B`

(`-c`, `-r`, `-u` modes)
Lets you specify the number of 512-byte blocks that fit on the media as one of the supplied *bBdfs-arg* values. This permits `tar` to correctly issue prompts requesting insertion of additional media during lengthy

archival operations.

bBdfs-arg
: (`-c, -r, -u` modes)
Specifies one of the arguments associated with the `-b`, `-B`, `-d`, `-f`, or `-s` command options. Note that these values are detached from their associated command options. However, they must be entered in an order that matches the order of the related command options. For more information regarding legal replacement values, refer to each of the specific command options elsewhere in this alphabetical list.

`-c` Creates a new archive, overwriting any old information within an archive.

`-d`
: (`-c, -r, -u` modes)
Lets you specify the tape's density as one of the supplied *bBdfs-arg* values. The default value is 1600 (BPI).

`-f` Lets you specify the archive file or device as one of the supplied *bBdfs-arg* values. The default value is `/dev/mt/0m`.

If the value is given as `-`, `tar` writes to the standard output or reads from the standard input, in accordance with other command options. Thus, `tar` may be used at the head or the tail of a command pipeline as shown in ''Examples.''

file
: (`-c, -r, -u` modes)
Specifies what to archive in terms of a file or directory. If a directory is specified, the contents of it (including nested directories and files) are archived. Note that `tar` does not follow symbolic links properly, but that it does correctly handle hard file links.

file-in-archive
: (`-t` mode)
Specifies what to list from an archive in terms of one or more exactly-matching filenames (or pathnames) that you know are located inside an archive and that you wish to be displayed (see the other options for use when retrieving files from archives, `-m` and `-o`).

file-in-archive
: (`-x` mode)
Specifies what to retrieve from an archive in terms of one or more exactly-matching filenames (or pathnames) of files that are located inside an archive and that you wish to retrieve. If the file named matches a directory whose contents had been written into *archive*, this directory is (recursively) extracted. If a named file from within *archive* does not exist on the system, the file is created with the same

mode as in the archive, except that the set-user-ID and set-group-ID bits are not set unless you are superuser. If the file exists, its permissions are not changed except for the bits described above. The owner, group, and modification time are restored (if possible). If no *file-in-archive* arguments are given, the entire content of *archive* is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.

`-i` Establishes the preference to ignore symbolic links (see related information within ''Limitations'').

`-l`

(`-c`, `-r`, `-u` modes)
Establishes the preference to report error messages if links cannot be followed, causing certain data to be skipped. If `-l` is not specified, no error messages are printed.

`-m`

(`-x` mode only)
Establishes new modification times for files retrieved from an archive rather than preserving the original modification dates and times. The modification time of the file will be the time of extraction.

`-o`

(`-x` mode only)
Establishes the user ID and group ID for retrieved files as those of the user running the program rather than those recorded when the archive was created.

`-r` Appends archive data onto an existing archive rather than overwriting and destroying the previous archive. This command mode may not work on all devices because it requires the ability to perform ''seek'' operations within the media.

`-s` Lets you specify the tape's length in feet as one of the supplied *bBdfs-arg* values. This value together with the density allows `tar` to calculate when the end of the tape is reached, so that it can prompt for another tape in a sequence of tapes. The default length is 2300 feet.

`-t` Displays the contents of an archive without copying any files out of it.

`-u` Appends to an archive conditionally, adding files to the archive file or device (see the `-f` option) only if they are not already there or have been modified since last written on that archive.

`-v` Establishes a preference to provide regular feedback about the work underway. Normally, `tar` does its work without displaying progress messages. The `-v` (verbose) option causes it to type the name of each file it treats, preceded by the function letter. When used along with the `-t` option, `-v` gives, in addition to the name, information about the tape entries.

`-w`
: (`-t`, `-x` modes)
Requests confirmation between individual read or write operations. When this command option is present, `tar` prints the action to be taken, followed by the name of the file, and then waits for the user's confirmation. If a word beginning with `y` is given, the action is performed. Any other input means ''no''.

`-x` Retrieves (extracts) files from an archive file or device.

DESCRIPTION

`tar` saves and retrieves files within an archive, which is often stored on removable media such as magnetic tape or 3.5-inch disks. Archives come in two basic varieties. Archives that are not embedded inside of files are called archives. They can occupy all of the available storage space. Archives that are embedded within files inside of file systems are called archive files. They can occupy the remaining storage once the overhead for the file system has been subtracted from the disk or tape media. The terms archive and `tar`-formatted data are not specific with regard to a storage format.

When placed inside a normal file, an archive acquires all the usual file attributes, such as a filename and permission settings. When archive files are created, the file attributes are those associated with Macintosh files if the archive file is copied to a Macintosh disk using the Finder, or UNIX file attributes if left to reside on an A/UX file system.

Five possible modes of operation are selected through the leading command option. Of these, two perform read operations, and three perform a variety of write operations. Use the `-x` option to read from a tar-formatted stream of data. The following command line uses `tar` to read from the 3.5-inch disk inside drive zero and retrieve all of its `tar`-formatted files.

```
tar -xf /dev/rfloppy0
```

The `-c` option creates a stream of `tar`-formatted data containing a copy of all the files supplied as *file* arguments. To create an archival version of each of the files `cover`, `body`, and `graph` inside of the archive file named `project.arc`, use

```
tar -cf project.arc  cover body graph
```

EXAMPLES

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

will copy directories from one directory tree (fromdir) to another (todir).

STATUS MESSAGES AND VALUES

If tar reports that not enough memory is available to hold the link tables, then the archive will not reliably maintain information about which files are linked.

LIMITATIONS

There is no way to ask for the *n*th occurrence of a file inside a given archive.

Tape errors are handled ungracefully.

The -b option should not be used with archives that are going to be updated. The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the -b option should not be used at all, because updating an archive stored on disk can destroy it.

The current limit on filename length is 100 characters.

Empty directories are skipped when creating a tar archive.

Note that tar c0m is not the same as tar cm0.

tar is unable to create archives containing device files.

FILES

```
/usr/bin/tar
/dev/rmt/*
/dev/mt/*
/tmp/tar*
```

SEE ALSO

ar(1), cpio(1), dd(1), tp(1)

dump.bsd(1M) in *A/UX System Administrator's Reference*

tar(4) in *A/UX Programmer's Reference*

NAME

`tbl` — table formatter for `nroff` or `troff`

SYNOPSIS

`tbl [-TX]` [*file*]...

ARGUMENTS

file Specifies the file (that contains a table) to be formatted. If no filenames are given as arguments (or if `-` is specified as the last argument), `tbl` reads the standard input, so it may be used as a filter. When it is used with `eqn`(1) or `neqn`, `tbl` should come first to minimize the volume of data passed through pipes.

`-TX`
Forces `tbl` to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (for example, line printers).

DESCRIPTION

`tbl` is a preprocessor that formats tables for `nroff` or `troff`. The input files are copied to the standard output, except for lines between `.TS` and `.TE` command lines, which are assumed to describe tables and are reformatted by `tbl`. (The `.TS` and `.TE` command lines are not altered by `tbl`.)

The `.TS` command line is followed by global options, which are:

`center`
Centers the table (default is left adjust).

`expand`
Makes the table as wide as the current line length.

`box`
Encloses the table in a box.

`doublebox`
Encloses the table in a double box.

`allbox`
Encloses each item of the table in a box.

`tab(`*x*`)`
Uses the character *x* instead of a tab to separate items in a line of input data.

The global options, if any, are terminated with a semicolon (`;`).

Next come lines describing the format of each line of the table. Each such format line describes one line of the actual table, except that the last format line (which must end with a period) describes *all* remaining lines of the table. Each column of each line of the table is described by a single keyletter, optionally followed by specifiers that determine the font and point size of the corresponding item, that indicate where vertical bars are to appear between columns, that determine column width, intercolumn spacing, etc. The available keyletters are:

`c`	Centers item within the column.
`r`	Right adjusts item within the column.
`l`	Left adjusts item within the column.
`n`	Numerically adjusts item in the column. Units positions of numbers are aligned vertically.
`s`	Spans previous item on the left into this column.
`a`	Centers longest line in this column and then left adjusts all other lines in this column with respect to that centered line.
`^`	Spans down previous entry in this column.
`_`	Replaces this entry with a horizontal line.
`=`	Replaces this entry with a double horizontal line.

The characters `B` and `I` stand for the bold and italic fonts, respectively. The character `|` indicates a vertical line between columns.

The format lines are followed by lines containing the actual data for the table, followed finally by `.TE`. Within such data lines, data items are normally separated by tab characters.

If a data line consists of only _ or =, a single or double line, respectively, is drawn across the table at that point. If a single item in a data line consists of only _ or =, then that item is replaced by a single or double line.

Full details of all these and other features of `tbl` are given in the reference manual cited below.

EXAMPLES

In the following input, CONTROL-I (`^I`) represents a tab (which should be typed as a genuine tab):

```
.TS
center box ;
cB s s
cI cI s
cI | cI s
l | n n .
Household | Population
_
Town^IHouseholds
^INumber^ISize
```

```
=
Bedminster^I789^I3.26
Bernards Twp.^I3087^I3.74
Bernardsville^I2018^I3.30
Bound Brook^I3425^I3.04
Bridgewater^I7897^I3.81
Far Hills^I240^I3.19
.TE
```

yields:

Household Population		
Town	*Households*	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

LIMITATIONS

See LIMITATIONS under `nroff`(1).

FILES

`/bin/tbl`
Executable file

SEE ALSO

`eqn`(1), `mm`(1), `mvt`(1), `nroff`(1), `troff`(1)

`mm`(5), `ms`(5), `mv`(5) in *A/UX Programmer's Reference*

``tbl` Reference'' in *A/UX Text Processing Tools*

NAME

`tc` — interprets troff output for use at a vintage display device

SYNOPSIS

`tc` [`-a` *n*] [`-e`] [`-o` *list*] [`-t`] [*file*]...

ARGUMENTS

`-a` *n*
: Sets the aspect ratio to *n*; the default is 1.5.

`-e` Does not erase before each page.

`-o` *list*
: Prints only the pages enumerated in *list.* The list consists of pages and page ranges (for example, 5-17) separated by commas. The range *n*- goes from *n* to the end; the range -*n* goes from the beginning to and including page *n*.

`-t` Does not wait between pages (for directing output into a file).

DESCRIPTION

`tc` interprets its input (standard input default) as output from `troff`(1). The standard output of `tc` is intended for a TEKTRONIX 4015 (a 4014 terminal with ASCII and APL character sets). The various typeface sizes are mapped into the 4014's four sizes; the entire `troff` character set is drawn with the 4014's character generator, by using overstruck combinations where necessary, producing an altogether displeasing effect.

Typical usage is:

`troff` *troff-options file* `| tc`

At the end of each page, `tc` waits for a newline (empty line) from the keyboard before continuing to the next page. In this wait state, the following commands are recognized:

`!`*cmd*
: Sends *cmd* to the shell.

`e` Inverts the state of the screen erase.

n Prints page *n* (previously printed).

-*n* Skips backward *n* pages.

`o`*list*
: Sets `-o` list to *list.*

`p` Prints current page again.

`a`*n* Sets the aspect ratio to *n*.

`?` Prints list of available options.

LIMITATIONS

Font distinctions are lost.

The `tc` command needs a `-w` option to wait for input to arrive.

FILES

`/usr/bin/tc`
: Executable file

SEE ALSO

`4014`(1), `nroff`(1), `tplot`(1G), `troff`(1)

NAME

tcb — blocks data to 8K for direct input to /dev/rmt/tc*x*

SYNOPSIS

command-line | tcb >/dev/rmt/tc*x*

ARGUMENTS

command-line
Specifies the command line which generates the output channeled to the tcb command.

x Specifies the Small Computer System Interface (SCSI) ID of the tape cartridge drive.

DESCRIPTION

tcb reads standard input and writes standard output in a blocking format suitable for the Apple Tape Backup 40SC. The output of tcb is always blocked at 8K to satisfy the blocking requirements of the tape cartridge drive. The last output block is zero-filled as necessary. If the output of tcb is sent through another pipe before being directed to the tape cartridge drive, the tape-specific blocking of tcb data is lost.

To create an archive that is larger than one tape, specify tar along with the -B option and a size argument. This option allows you to specify the capacity of the tape cartridge in terms of 512-byte blocks. You compute the capacity by using a conversion factor of 2048 blocks per MB.

To read from an archive written with tar and tcb, use dd(1) and specify an 8K blocking size. Better performance results from using a larger input buffer size, as long as the buffer size is a multiple of 8K:

```
dd if=/dev/rmt/tcx ibs=20x8K |tar xvf -
```

EXAMPLES

The following example illustrates how to create a tar archive on cartridge tape. The SCSI ID of your tape cartridge drive is 2 in this example. Type:

```
tar cvf - . | tcb >/dev/rmt/tc2
```

SEE ALSO

dd(1), tar(1)

tc(7) in *A/UX Programmer's Reference*

NAME

`tee` — transcribes data

SYNOPSIS

`tee` [`-i`] [`-a`] [*file*]...

ARGUMENTS

`-a` Causes the output to be appended to the *files* rather than overwriting them.

file Specifies the file that contains the transcribed data.

`-i` Ignores interrupts.

DESCRIPTION

`tee` transcribes the standard input to the standard output and makes copies in the specified files.

EXAMPLES

The command:

```
make | tee x
```

will cause the output of the *make* program to be recorded on file `x` as well as printed on standard output.

FILES

`/bin/tee`
Executable file

NAME

`telnet` — communicates with another host via the TELNET protocol

SYNOPSIS

`telnet` *host* [*port*]

`telnet`

ARGUMENTS

host [*port*]
: Specifies the name of the host system you wish to connect to, and the port number of that host.

DESCRIPTION

`telnet` is used to communicate with another host using the TELNET protocol.

If `telnet` is invoked with arguments, it performs an `open` command (see the ``open'' command below) with those arguments.

If `telnet` is invoked without arguments, it enters command mode, indicated by its prompt (``telnet>''). In this mode, it accepts and executes the commands listed below.

Once a connection has been opened, `telnet` enters an input mode. The input mode entered will be either ``character at a time'' or ``line by line'' depending on what the remote system supports.

In ``character at a time'' mode, most text typed is immediately sent to the remote host for processing.

In ``line by line'' mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The ``local echo character'' (initially ``^E'') may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the *localchars* toggle is TRUE (the default in line mode; see below), the user's quit, interrupt, and flush characters are trapped locally, and sent as TELNET protocol sequences to the remote side. There are options (see `toggle` *autoflush* and `toggle` *autosynch* below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of quit and interrupt).

While connected to a remote host, `telnet` command mode may be entered by typing the `telnet` ``escape character'' (initially ``^]''). When in command mode, the normal terminal editing conventions are available.

Commands

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the `mode`, `set`, `toggle`, and `display` commands).

`open` *host* [*port*]
: Opens a connection to the named *host*. If no *port* number is specified, `telnet` attempts to contact a TELNET server at the default port. The host specification may be either a host name (see `hosts`(4)) or an Internet address specified in the ''dot notation'' (see `inet`(3N)).

`close`
: Closes a TELNET session and returns to command mode.

`quit`
: Closes any open TELNET session and exits `telnet`. An end of file (in command mode) will also close a session and exit.

`z` Suspends `telnet`. This command only works when the user is using the `csh`(1).

`mode` *type*
: Sets *type* to either *line* (for ''line by line'' mode) or *character* (for ''character at a time'' mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

`status`
: Shows the current status of `telnet`. This includes the peer one is connected to, as well as the current mode.

`display` [*argument*]...
: Displays all, or some, of the `set` and `toggle` values (see below).

`?`[*command*]
: Gets help. With no arguments, `telnet` prints a help summary. If a command is specified, `telnet` prints the help information for just that command.

`send` *arguments*
: Sends one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

escape
: Sends the current `telnet` escape character (initially ''`^]`'').

synch
: Sends the TELNET SYNCH sequence. This sequence causes the remote system to discard all previously typed (but not yet read)

input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system; if it doesn't work, a lowercase ''r'' may be echoed on the terminal).

brk Sends the TELNET BRK (Break) sequence, which may have significance to the remote system.

ip Sends the TELNET IP (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

ao Sends the TELNET AO (Abort Output) sequence, which should cause the remote system to flush all output *from* the remote system *to* the user's terminal.

ayt Sends the TELNET AYT (Are You There) sequence, to which the remote system may or may not choose to respond.

ec Sends the TELNET EC (Erase Character) sequence, which should cause the remote system to erase the last character entered.

el Sends the TELNET EL (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

ga Sends the TELNET GA (Go Ahead) sequence, which likely has no significance to the remote system.

nop Sends the TELNET NOP (No OPeration) sequence.

? Prints out help information for the `send` command.

`set` *argument value*

Set any one of a number of `telnet` variables to a specific value. The special value ''off'' turns off the function associated with the variable. The values of variables may be interrogated with the `display` command. The variables which may be specified are:

echo

This is the value (initially ''`^E`'') which, when in ''line by line'' mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

escape

This is the `telnet` escape character (initially ''`^[`'') which causes entry into `telnet` command mode (when connected to a remote system).

interrupt

If `telnet` is in *localchars* mode (see `toggle` *localchars* below) and the interrupt character is typed, a TELNET IP sequence (see `send` *ip* above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's interrupt character.

quit If `telnet` is in *localchars* mode (see `toggle` *localchars* below) and the quit character is typed, a TELNET BRK sequence (see `send` *brk* above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's quit character.

flushoutput
If `telnet` is in *localchars* mode (see `toggle` *localchars* below) and the *flushoutput* character is typed, a TELNET AO sequence (see `send` *ao* above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's flush character.

erase
If `telnet` is in *localchars* mode (see `toggle` *localchars* below), *and* if `telnet` is operating in ''character at a time'' mode, then when this character is typed, a TELNET EC sequence (see `send` *ec* above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's erase character.

kill If `telnet` is in *localchars* mode (see `toggle` *localchars* below), `and` if *telnet* is operating in ''character at a time'' mode, then when this character is typed, a TELNET EL sequence (see `send` *el* above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's kill character.

eof If *telnet* is operating in ''line by line'' mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the EOF character is taken to be the terminal's EOF character.

`toggle` *arguments...*
Toggles (between TRUE and FALSE) various flags that control how `telnet` responds to events. More than one argument may be specified. The state of these flags may be interrogated with the `display` command. Valid arguments are:

localchars
If this is TRUE, then the flush, interrupt, quit, erase, and kill characters (see `set` above) are recognized locally, and transformed into (hopefully) appropriate TELNET control sequences (respectively *ao*, *ip*, *brk*, *ec*, and *el*; see `send` above). The initial value for this toggle is TRUE in ''line by line'' mode, and FALSE in ''character at a time'' mode.

autoflush

If *autoflush* and *localchars* are both TRUE, then when the *ao*, interrupt or quit characters are recognized (and transformed into TELNET sequences; see `set` above for details), `telnet` refuses to display any data on the user's terminal until the remote system acknowledges (via a TELNET Timing Mark option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user had not done an `stty noflsh`, otherwise FALSE (see `stty(1))`.

autosynch

If *autosynch* and *localchars* are both TRUE, then when either the interrupt or quit characters is typed (see `set` above for descriptions of the interrupt and quit characters), the resulting TELNET sequence sent is followed by the TELNET SYNCH sequence. This procedure *should* cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

crmod

Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a linefeed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never linefeed. The initial value for this toggle is FALSE.

debug

Toggles socket level debugging (useful only to the superuser). The initial value for this toggle is FALSE.

options

Toggles the display of some internal `telnet` protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.

netdata

Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

? Displays the legal `toggle` commands.

LIMITATIONS

There is no adequate way for dealing with flow control.

On some remote systems, echo has to be turned off manually when in ''line by line'' mode.

There is enough settable state to justify a `.telnetrc` file.

No capability for a `.telnetrc` file is provided.

In ''line by line'' mode, the terminal's EOF character is only recognized (and sent to the remote system) when it is the first character on a line.

FILES

`/usr/bin/telnet`
: Executable file

SEE ALSO

`cu`(1C), `ftp`(1N), `rlogin`(1N), `tip`(1C), `uucp`(1C)

`telnetd`(1M) in *A/UX System Administrator's Reference*

NAME

`test` — evaluates conditions

SYNOPSIS

`test` [*expr*]

ARGUMENTS

expr
: Specifies the expression to be evaluated by the `test` command.

DESCRIPTION

`test` evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a nonzero (false) exit status is returned. The `test` command also returns a nonzero exit status if there are no arguments.

The superuser is always granted execute permission even though execute permission is meaningful only for directories and regular files and `exec` requires that at least one execute mode bit be set for a regular file to be executable. 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 device file.

`-b` *file*
: True if *file* exists and is a block device 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 string *s1* is nonzero.

s1 `=` *s2*
: True if strings *s1* and *s2* are identical.

s1 `!=` *s2*
: True if strings *s1* and *s2* are *not* identical.

s1 True if *s1* is *not* the null string.

n1 `-eq` *n2*
: True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons `-ne`, `-gt`, `-ge`, `-lt`, and `-le` may be used in place of `-eq`.

These primaries may be combined with the following operators:

`!` Unary negation operator.

`-a` Binary AND operator.

`-o` Binary OR operator (`-a` has higher precedence than `-o`).

`(`*expr*`)`
: Parentheses for grouping.

Notice that all the operators and flags are separate arguments to `test`. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

EXAMPLES

The `test` command is typically used in shell scripts (`sh`(1)), as follows:

```
if test -d foo
then
        echo "foo is a directory"
fi
```

The following message:

```
foo is a directory
```

will print if it is found to be one when `test` is run.

FILES

`/bin/test`
: Executable file

SEE ALSO

`find`(1), `ksh`(1), `sh`(1)

''Bourne Shell Reference'' in *A/UX Shells and Shell Programming*

''Korn Shell Reference'' in *A/UX Shells and Shell Programming*

NAME

TextEditor — lets you edit files interactively through mouse and menu operations

SYNOPSIS

TextEditor [*file*]...

ARGUMENTS

file Specifies the file to be edited.

DESCRIPTION

TextEditor is a mouse-based text-editing program for use with both Macintosh and A/UX text-only files. It is compatible with A/UX 2.0 and later systems. TextEditor provides an alternative to the vi and ed text editors for those who prefer to work with the mouse and pull-down menus rather than with keyboard commands.

You can start TextEditor by double-clicking its icon, by double-clicking the icon of a Macintosh or A/UX text file while TextEditor is the default editor, or by typing TextEditor on the A/UX command line. (For the last two launch methods to work, your startup file should set the search directory list stored in PATH to include /mac/bin, and FINDER_EDITOR should be set to /mac/bin/TextEditor.) For more information about making TextEditor the default editor, see *A/UX Essentials.*

If you double-click a file icon or specify a filename when invoking TextEditor from the command line, the text of that file appears in the first window displayed; otherwise, the first window is empty. TextEditor lets you open several windows at once, each displaying text from a different file; however, you can work in only one window at a time. The window in which you are working is called the ''active window.''

You can scroll and page through the text in the active window by using the scroll bar that appears at its right side, as described in *A/UX Essentials*.

Files created or edited by TextEditor are saved as text-only files of type 'TEXT'. They can contain tab and newline characters but no other formatting information. This file structure is compatible with that of other applications that create text-only files; for example, TextEditor can process MacWrite files saved with the Text Only option.

The tab setting, font setting, selection, window settings, auto-indent state, invisible character state, and markers applicable to a file are saved with the file in its resource fork. This resource fork appears as a file named %*file* in the A/UX directory that contains the primary file. You can tell TextEditor not to save this resource file by clicking the Save Text Only radio button in the dialog box that appears when you select any of the following items from the File menu: New, Close, Save As, and Save a

Copy.

Mouse-Based Editing

In `TextEditor`, the procedure for inserting text entered from the keyboard is simple. Use the mouse to position the I-beam pointer on the screen at any place in the text inside the text window and then click (press and release) the mouse button. When you click, a blinking vertical bar appears at the pointer position to mark the current text-insertion point. Characters you enter from the keyboard always appear at this insertion point. At any time, you can move the pointer to a new place in the text and click to establish a new insertion point.

> *Caution:* Except for the tab and newline characters, `TextEditor` ignores zero-width (control) characters generated by the keyboard. If you need to enter such a character into a document, generate it in the Key Caps desk accessory (accessible in the Apple menu) and use the Copy and Paste menu items in the Edit menu to transfer it to the document.

The general procedure for using `TextEditor` to edit or otherwise modify existing text comprises two steps: First you select the text to be changed and then you choose the operation you want to perform on the selection.

If you select text and immediately enter one or more characters from the keyboard, rather than choosing a menu item, `TextEditor` deletes the selected text and inserts the text entered from the keyboard in its place.

In many cases, `TextEditor` lets you undo an operation if you make a mistake. Choose Undo from the Edit menu immediately after the faulty operation.

The next section, ''Text Selection,'' tells you how to select a range of text. The subsequent section, ''Menu Items,'' describes the operations you can perform.

Text Selection

You can use several techniques to select a range of text for a `TextEditor` editing operation.

Double-clicking. When you position the pointer on a word and click the mouse button twice in rapid succession, `TextEditor` selects that word. This technique is called ''double-clicking.'' In this selection mode, `TextEditor` recognizes two character domains. One domain contains the uppercase and lowercase letters, the ten numerals, and the underscore character; the other domain contains all other characters, including punctuation, space, and newline. If you double-click a character from the letter domain, `TextEditor` selects text in both directions from that character to the first character belonging to the punctuation domain. If you double-click on a punctuation character, except for one of the enclosing characters described later in ‘‘Enclosed Text Selection,’’ `TextEditor` selects just that character.

Triple-clicking. When you place the cursor anywhere within a line of text and click the mouse button three times in rapid succession, `TextEditor` selects the entire line. This technique is called ‘‘triple-clicking.’’

Dragging. When you move the pointer over text from one place to another while holding down the mouse button, `TextEditor` selects all the text the pointer passes over until you release the mouse button. This technique is called ‘‘dragging.’’ By dragging, you can select any amount of text from a single character to an entire document. When you attempt to move the pointer above or below the text currently displayed, `TextEditor` automatically scrolls the window to display more text.

Shift selection. When you move the pointer to a place other than the current insertion point and then click while holding down the SHIFT key, `TextEditor` selects all the text between the insertion point and the pointer position, even when they are on different pages of the document and the insertion point is not visible.

Marker selection. You can create names for selections of text as described in ‘‘Mark Menu’’ in the ‘‘Menu Items’’ section, later in the ‘‘Description’’ section. To select a piece of text that you have previously named, you just choose its name from the Mark menu.

Enclosed-text selection. When you double-click on particular delimiters, `TextEditor` selects all text between the delimiter you double-clicked and its mating delimiter. The delimiter characters and their mates are

(*text*)
[*text*]
{*text*}

This method of selecting text works both backward and forward. For example, if you double-click on a right bracket, `TextEditor` searches backward for the first preceding left bracket. It also correctly parses nested structures that use the same enclosing characters.

When you double-click the first occurrence of a delimiter in the following list, `TextEditor` forward-selects all text between it and the next occurrence of the same delimiter. The self-mating delimiters are

" ' ` / \

When you make a selection by double-clicking any of the delimiter characters just listed, `TextEditor` selects all characters between that delimiter and the mating delimiter. The search for the mating character continues until the beginning or end of the document is reached. The resulting selection does not include the delimiters themselves. If `TextEditor` does not find a mating delimiter, it selects only the character that you originally double-clicked.

Menu Items

`TextEditor` displays menus titled File, Edit, Find, Mark, and Window in the menu bar at the top of the screen, plus the Apple menu at the far left. Each menu contains menu items that perform various `TextEditor` actions.

To choose a menu item, position the pointer on a menu title, press the mouse button, and move the mouse downward while holding down the mouse button. Release the button when the pointer has highlighted the desired item. Menu actions operate only on the active window.

You can invoke many menu actions from the keyboard by holding down the COMMAND key (not CONTROL) while typing a character. The character required is shown beside the COMMAND-key symbol in the menu display. Such COMMAND-key equivalents can be entered with lowercase characters; you don't need to hold down the SHIFT key as well.

The sections that follow describe the actions performed by the various `TextEditor` menu items.

Apple menu. At the far left of the menu bar, the Apple symbol is the title of a menu that contains the About TextEditor menu item. Choosing that menu item displays a dialog box that gives version information.

File menu. The menu items in the File menu let you create, retrieve, and save files, print text, and quit `TextEditor`. The File menu contains the following menu items:

New...

Creates a new empty file of type 'TEXT'. This menu item first displays a dialog box that lets you enter a filename and select a directory to contain the document. When you click the Desktop button in this dialog box, `TextEditor` lists the names of disks (volumes) that appear on the desktop, to allow you to switch between them. Once you have selected a volume, and possibly a folder or nested folder on a volume, you can click the New button to create an empty file in the selected location. At that time, a new, active `TextEditor` window appears.

The dialog box contains radio buttons that let you specify whether the resulting file will be saved with its formatting information or as text only.

The COMMAND-key equivalent for the New menu item is COMMAND-N.

Open...

Opens an existing text file from a disk. This menu item first displays a list of all files of type 'TEXT' that are available in the current directory. You can click the Desktop button to view the names of disks (volumes) that appear on the desktop. To open a file, double-click its name, or select its name and then click the Open button. When you open a file for the first time, `TextEditor` places the insertion point at the beginning of the text. When you open the file subsequently, it appears in the last state in which `TextEditor` saved it; the previous selection or insertion point is preserved (if you have saved formatting information) unless the file has been modified by other software. To open a nonmodifiable copy of a file, click the Read-Only checkbox. If the file you specify is already open in `TextEditor`, its window is made active. The COMMAND-key equivalent for the Open menu item is COMMAND-O.

Close

Closes the active window and removes it from view. You can display the window later by using the Window menu. This menu item does not save the window contents on a disk.

If you have not previously saved the file, a dialog box appears that lets you specify whether the file being closed is to be saved and, if so, whether it should be saved with formatting information or as a text-only file. The COMMAND-key equivalent for the Close menu item is COMMAND-W.

Save

Saves the contents of the active window on a disk, in the file that was originally opened, without closing the window. This menu item is dimmed if the contents of the file have not been changed since the last Save action. The COMMAND-key equivalent for the Save menu item is COMMAND-S.

Save As...

Allows you to make a copy of the currently active file, which you must then save under a different name. This menu item saves the current contents of the window as the new file, and allows you to continue editing the new file. The old file is left unchanged with its original name.

Save a Copy...

Allows you to save the current state of the active window to a new file with the name ''Copy Of *file*.''

Revert to Saved
: Discards all changes that you have made to the contents of the active window since they were last saved. This menu item is dimmed if the contents of the file have not been changed since the last Save action.

Page Setup
: Displays a dialog box that lets you set the page size, orientation, and reduction or enlargement for subsequent printing actions.

Print Window
: Prints text from the active window. If part of the text is selected, `TextEditor` prints only the selection; otherwise, it prints the entire document displayed in the window. Use the Chooser desk accessory, available in the Apple menu, to specify which printer to use. Use the Page Setup menu item to specify page size, orientation, and scale.

Quit
: Quits `TextEditor` and returns you to the Finder. If there are unsaved changes to any files, `TextEditor` gives you a chance to save them. The COMMAND-key equivalent for the Quit menu item is COMMAND-Q.

Edit menu. The menu items in the Edit menu allow you to move text and perform global formatting actions. The Edit menu contains the following menu items:

Undo
: Reverses the most recent text change. If you choose Undo a second time, the change is reinstated. This menu action does not affect changes to the resource fork, such as font or tab settings. If there was no previous action taken, this menu item is dimmed. The COMMAND-key equivalent for the Undo menu item is COMMAND-Z.

Cut Removes the currently selected text from the active window and places it in the Clipboard. The COMMAND-key equivalent for the Cut menu item is COMMAND-X.

Copy
: Copies the currently selected text in the active window to the Clipboard without deleting it from the window. The COMMAND-key equivalent for the Copy menu item is COMMAND-C.

Paste
: Replaces the currently selected text in the active window with the contents of the Clipboard. If there is no current selection, `TextEditor` inserts the contents of the Clipboard at the current insertion point. The COMMAND-key equivalent for the Paste menu item is COMMAND-V.

Clear
: Deletes the currently selected text from the active window. The keyboard equivalent for the Clear menu item is DELETE.

Select All
: Selects the entire document that is displayed in the active window. The COMMAND-key equivalent for the Select All menu item is COMMAND-A.

Show Clipboard
: Opens a new, active `TextEditor` window that displays the contents of the Clipboard, if any.

Format...
: Displays a dialog box that lets you set typography and indentation for the entire document that is displayed in the active window.

  The scroll box lets you select a font and size for the active window by clicking items in the lists.

  The Auto Indent checkbox in the Format dialog box turns auto-indenting on and off. When auto-indenting is on, press RETURN to align text with the left margin of the preceding line. You can override auto-indenting for any single line, aligning it to the far left margin, by holding down OPTION while you press RETURN.

  The Show Invisibles checkbox in the Format dialog box turns the display mode for invisible characters on and off. When display mode is on, all characters in the document are displayed, including those normally invisible. Tabs are shown as triangles, spaces as diamonds, newlines as logical negation characters (¬), and all other normally invisible characters as upside-down question marks.

  The Tabs text box in the Format dialog box lets you enter the number of spaces signified by each tab character in the active window.

Align
: Aligns the left margin of all the currently selected text in the active window with the top line of the selection.

Shift Left
: Moves the currently selected text in the active window one tab measure to the left, preserving indentation within the selection. The COMMAND-key equivalent for the Shift Left menu item is COMMAND-{. If you also hold down the SHIFT key while pressing COMMAND-{, the distance moved becomes one space instead of one tab.

Shift Right
: Performs the same action as Shift Left, but moves the selection to the right. The COMMAND-key equivalent for the Shift Right menu item is

COMMAND-}.

Find menu. The menu items in the Find menu help you find and replace text in the active window.

All search actions start by displaying a dialog box that lets you specify the following options by clicking a checkbox:

Literal
: Finds the exact string you entered, wherever it may appear, even if it is part of another string.

Entire Word
: Finds the string you entered only if it constitutes an entire word. The determination of word boundaries is the same as with double-clicking, described in ''Text Selection,'' earlier in the ''Description'' section. The Entire Word and Literal options are mutually exclusive.

Case Sensitive
: Finds the string you entered only if the uppercase and lowercase status of all letters in the found string is the same as that of the letters in the search string.

Search Backward
: Searches from the current selection or insertion point toward the beginning of the document. You can temporarily reverse the direction of searching, either from forward to backward or from backward to forward, by holding down the SHIFT key when you start a search operation.

Wrap-around Search
: Searches forward to the end of the document, then starts searching again from the beginning to the current selection or insertion point. If Search Backward is also selected, Wrap-around Search does the same operation in the reverse direction.

By default, only the Literal checkbox is selected. Whenever a search fails, `TextEditor` produces a beep, or other sound you set as your preferred alert sound.

The Find menu contains the following menu items:

Find...
: Finds the next occurrence of the string you specify in the text box. `TextEditor` scrolls the active window to display the found string and selects the text it has found. The COMMAND-key equivalent for the Find menu item is COMMAND-F.

Find Same
: Repeats the most recent Find operation. The COMMAND-key

equivalent for the Find Same menu item is COMMAND-G.

Find Selection
: Finds the next occurrence of the currently selected text. The COMMAND-key equivalent for the Find Selection menu item is COMMAND-H.

Display Selection
: Scrolls the active window to display the currently selected text.

Replace...
: Finds the next occurrence of the string you specify in the text box and replaces it with another string that you also specify in a second text box. The COMMAND-key equivalent for the Replace menu item is COMMAND-R.

Replace Same
: Repeats the latest Replace operation. The COMMAND-key equivalent for the Replace Same menu item is COMMAND-T.

Mark menu. The menu items in the Mark menu help you navigate long documents. They let you associate labels with pieces of text so you can easily find them later. They also make it easy to select large pieces of text, as explained in ''Text Selection,'' earlier in the ''Description'' section.

The upper part of the Mark menu, above the horizontal line, contains the menu items Mark and Unmark; the lower part contains a list of all mark labels that you have created for the currently active window. The Mark and Unmark menu items perform the following actions:

Mark...
: Displays a dialog box that lets you attach a label to a text position. If you previously selected a piece of text, the label applies to the whole selection; if not, it applies to the current position of the insertion point. If you try to create a label by using a name that is already taken, `TextEditor` displays a dialog box that lets you either replace the old marker or choose a new name. The COMMAND-key equivalent for the Mark menu item is COMMAND-M.

Unmark...
: Displays a dialog box that lets you remove unwanted markers. The Unmark dialog box shows you a list of all current markers. You can select one or more of them, by clicking or dragging, and then click the Delete button. If you decide you don't want to delete a marker, click the Cancel button.

When you choose one of the label items in the lower part of the menu, `TextEditor` scrolls the active window to the marked text and either selects it (if you originally marked a selection) or places the insertion point at the marked position.

Window menu. The menu items in the Window menu allow you to arrange and display `TextEditor` windows. The upper part of the Window menu, above the horizontal line, contains the menu items Tile Windows and Stack Windows; the lower part contains a list of the absolute pathnames of all open `TextEditor` windows.

The menu items in the upper part of the Window menu perform the following actions:

Tile Windows
: Arranges the currently open windows vertically, so that at least part of the contents of each is visible.

Stack Windows
: Arranges the currently open windows in a diagonally staggered overlapping pattern, with the active window in front. The active window is the only one whose contents are visible.

When you choose one of the windows listed in the lower part of the menu, `TextEditor` makes it the active window. The names of currently displayed windows are listed in the order in which they were originally displayed. In addition, they are marked as follows:

Check symbol
: Indicates the currently active window.

Round bullet (•)
: Indicates the window that was active just before the currently active window, and hence is second to the front.

Underline (_)
: Indicates any window containing changes that have not yet been saved.

FILES

`/mac/bin/TextEditor`
: Executable file

SEE ALSO

`ed`(1), `ex`(1), `sed`(1), `vi`(1)

A/UX Essentials

A/UX Text Editing Tools

MPW 3.0 Reference

NAME

`tftp` — transfers files via the Trivial File Transfer Protocol (TFTP)

SYNOPSIS

`tftp` [*host*]

ARGUMENTS

host Causes `tftp` to use the specified remote host system (*host*) as the default host for future transfers (see the `connect` command below).

DESCRIPTION

`tftp` is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine.

Commands

Once `tftp` is running, it issues the prompt `tftp>` and recognizes the following commands:

`connect` *host-name* [*port*]

Sets the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers; thus, the `connect` command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the `connect` command; the remote host can be specified as part of the `get` or `put` commands.

`mode` *transfer-mode*

Sets the mode for transfers; The *transfer-mode* argument may be one of `ascii` or `binary`. The default is `ascii`.

`put` *file*
`put` *localfile remotefile*
`put` *file... remote-directory*

Puts a file or set of files to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host*:*filename* to specify both a host and filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers. If the *remote-directory* form is used, the remote host is assumed to be a UNIX machine.

`get` *filename*
`get` *remotename localname*
`get` *file...*

Gets a file or set of files from the specified *sources. source* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form to specify both a host and filename at the same time. If the latter form is used, the last hostname specified becomes the default for future transfers.

`quit`
: Exits `tftp`. An end of file also exits.

`verbose`
: Toggles verbose mode.

`trace`
: Toggles packet tracing.

`status`
: Shows current status.

`rexmt` *retransmission-timeout*
: Sets the per-packet retransmission timeout, in seconds.

`timeout` *total-transmission-timeout*
: Sets the total transmission timeout, in seconds.

`ascii`
: Causes `tftp` to set the mode to ‘‘mode ascii.’’

`binary`
: Causes `tftp` to set the mode to ‘‘mode binary.’’

`?` [*command-name*]...
: Prints help information.

LIMITATIONS

Because there is no user-login or validation within the TFTP protocol, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site and therefore difficult to document here.

FILES

`/usr/bin/tftp`
: Executable file

NAME

time — prints the elapsed time during the execution of a command

SYNOPSIS

time *command*

ARGUMENTS

command

Specifies the user-specified command that is being timed.

DESCRIPTION

time executes the *command* then 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 the standard error output.

EXAMPLES

The command:

```
time nroff -mm filea
```

will, in sh, perform the formatting and report the time at the end of the file, e.g.:

```
real  22.0
user   8.6
sys    6.4
```

In csh, where time is a built-in command, the time report might be:

```
8.9u  7.0s  0:29  54%
```

which reports, respectively, the user time, system time, real time, and percentage of real time that the CPU was active, which is the sum of the user and system times divided by the real elapsed time.

FILES

/bin/time

Executable file

SEE ALSO

csh(1), timex(1)

times(2) in *A/UX Programmer's Reference*

NAME

`timex` — reports the elapsed, user, and system time during the execution of a command

SYNOPSIS

`timex` [`-o`] [`-p[fhkmrt]`] [`-s`] *command*

ARGUMENTS

command
: Specifies the user-specified command that is sent to the shell for execution.

`-o` Reports the total number of blocks read or written and total characters transferred by *command* and all its children.

`-p[fhkmrt]`
: Lists process accounting records for *command* and all its children. The suboptions `f`, `h`, `k`, `m`, `r`, and `t` modify the data items reported, as defined in `acctcom`(1M). `timex` always reports the number of blocks read or written and the number of characters transferred.

`-s` Reports total system activity (not just that due to *command*) that occurred during the execution interval of *command*. The `timex` program reports all the data items listed in `sar`(1).

DESCRIPTION

`timex` sends the given *command* to the shell for execution, then reports (in seconds) the elapsed time, user time, and system time spent in execution. Optionally, `timex` may list or summarize process accounting data for the *command* and all its children, or report total system activity during the execution interval.

The output of `timex` is written on the standard error output.

EXAMPLES

This command:

```
timex ps -el
```

runs the `ps` command (with the correct options), then produces statistics concerning the command and system activity during the command to the standard error output.

WARNINGS

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 included spuriously.

FILES

`/usr/bin/timex`
Executable file

`/usr/lib/sa/timex`
File containing system activity information

SEE ALSO

`sar`(1), `time`(1)

`acctcom`(1M) in *A/UX System Administrator's Reference*

NAME

`tip` — establishes a connection to a remote system

SYNOPSIS

`tip` [`-v`] [`-`*speed]* *system-name*

`tip` [`-v`] [`-`*speed*] *phone-number*

ARGUMENTS

phone-number
: Specifies the telephone number of the modem that is connected to the remote system you wish to connect to.

speed
: Specifies the baud rate of the terminal. If *speed* is specified as 300 baud, but no system name is given, then `tip` assumes that a host with the name `tip300` exists in the `/etc/remote` file. Similarly, if this option is not given, but a *phone-number* is provided, then `tip` looks for a host with the name `tip0`. Refer to `remote`(4) for a full description.

system-name
: Specifies the name of the remote system you wish to connect to.

`-v` Causes `tip` to display the sets as they are made.

DESCRIPTION

`tip` establishes a full-duplex connection to another machine, giving the appearance of being logged in directly on the remote CPU. You must have a login (or equivalent) on the machine to which you wish to connect.

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well). A tilde (~) appearing as the first character of a line is an escape signal; the following are recognized.

`~.` Drops the connection and exit (you may still be logged in on the remote machine). You may also use ~CONTROL-D as a synonym for `~.`.

`~c`[*name*]
: Changes directory to *name* (no argument implies change to your home directory).

`~!` Escapes to a shell (exiting the shell will return you to `tip`).

`~>` Copies file from local to remote. `tip` prompts for the name of a local file to transmit.

`~<` Copies file from remote to local. `tip` prompts first for the name of the file to be sent, then for a command to be executed on the remote machine.

`~p` *from* [*to*]
: Sends a file to a remote UNIX host. The put command causes the remote UNIX system to run the command string `cat>'`*to*`'`, while `tip` sends it the *from* file. If the *to* file isn't specified, the *from* filename is used. This command is actually a UNIX-specific version of the `~>` command.

`~t` *from* [*to*]
: Takes a file from a remote UNIX host. As in the put command, the *to* file defaults to the *from* filename if the *to* file isn't specified. The remote host executes the command string

    `cat '`*from*`';echo` CONTROL-A

  to send the file to `tip`.

`~|` Pipes the output from a remote command to a local UNIX process. The `tip` command will prompt the user for both the remote command and the local command. The command string sent to the local UNIX system is processed by the shell. Note that the `eofread` variable should be set to the appropriate value before this escape is used.

`~#` Sends an interrupt signal to the remote system. For systems that do not support the necessary `ioctl` call, the break is simulated by a sequence of line speed changes and delete characters.

`~s` Sets a variable (see the discussion later in this section).

~CONTROL-Z
: Stops `tip` (available only with job control).

`~?` Gets a summary of the tilde escapes

The `tip` command uses the file `/etc/remote` to find how to reach a particular system and to find out how it should operate while talking to the system. Each system has a default baud with which to establish a connection. If this value is not suitable, the baud to be used may be specified on the command line, for example, `tip -300 mds`.

When `tip` establishes a connection, it sends out a connection message to the remote system; the default value, if any, is defined in `/etc/remote`.

The `tip` command also uses `/etc/dialup` to determine which modem escape sequences to use; refer to `remote`(4) and `dialup`(4) for details.

When `tip` prompts for an argument (for example, during setup of a file transfer), the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt or an interrupt will abort the dialogue and return to the remote machine.

The tip command guards against multiple users connecting to a remote system by opening modems and terminal lines with exclusive access, and by honoring the locking protocol used by uucp(1C).

During file transfers, tip provides a running count of the number of lines transferred. When using the ~> and ~< commands, the eofread and eofwrite variables are used to recognize end-of-file when reading, and specify end-of-file when writing (see ''Variables'' later in this section). File transfers normally depend on ixon/ixoff mode for flow control (see stty(1)). If the remote system does not support ixon/ixoff mode, echocheck may be set to indicate that tip should synchronize with the remote system on the echo of each transmitted character.

When tip must dial a telephone number to connect to a system, it will print various messages indicating its actions. tip supports the DEC DN-11 and Racal-Vadic 831 auto-call-units; the DEC DF02 and DF03, Ventel 212+, Racal-Vadic 3451, Bizcomp 1031 and 1032 integral call unit/modems, and Apple modems.

Variables

tip maintains a set of variables that control its operation. Some of these variables are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the s escape. The syntax for variables is patterned after vi(1) and mailx(1). Supplying all as an argument to the set command displays all variables readable by the user. Alternatively, the user may request display of a particular variable by attaching a ? to the end. For example, escape? displays the current escape character.

Variables are numeric, string, character, or boolean values. Boolean variables are set merely by specifying their names; they may be reset by prefixing a ! to the name. Other variable types are set by concatenating an = and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate, as well as set, a number of variables. Variables may be initialized at run time by placing set commands (without the ~s prefix in the file .tiprc in the user's home directory).

Certain common variables have abbreviations. Following is a list of common variables, with a description of each one, an abbreviation, and a default value (when applicable). The data type of each variable is listed in parentheses.

baudrate

(*num*) The baud at which the connection was established; abbreviated ba.

`beautify`
: (*bool*) Discard unprintable characters when a session is being scripted; abbreviated `be`.

`dialtimeout`
: (*num*) When dialing a telephone number, the time (in seconds) needed for a connection to be established; abbreviated `dial`.

`echocheck`
: (*bool*) Synchronize with the remote host during file transfer by waiting for the echo of the last character to be transmitted; default is `false`.

`eofcmd`
: (*str*) The string sent to indicate the end of remote command output (usually a prompt string) during a ~| pipe.

`eofread`
: (*str*) The set of characters which signify an end-of-transmission during a ~< file transfer command; abbreviated `eofr`.

`eofwrite`
: (*str*) The string sent to indicate end-of-transmission during a ~> file transfer command; abbreviated `eofw`.

`eol`
: (*str*) The set of characters which indicate an end-of-line. The `tip` program will recognize escape characters only after an end-of-line.

`escape`
: (*char*) The command prefix (escape) character; abbreviated `es`; default value is tilde (~).

`exceptions`
: (*str*) The set of characters which should not be discarded due to the beautification switch; abbreviated `ex`; default value is `\t\n\f\b`.

`force`
: (*char*) The character used to force literal data transmission; abbreviated `fo`; default value is CONTROL-P.

`framesize`
: (*num*) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated `fr`.

`halfduplex`
: (*bool*) Connection is half-duplex; abbreviated `hdx`. Default is `false`.

`host`
: (*str*) The name of the host connected to; abbreviated `ho`.

`localecho`
: (*bool*) Echo input locally; abbreviated `le`. Default is `false`.

`log`
: (*str*) The name of the file in which to log transaction activity reports. The default value is `/usr/adm/aculog`.

`prompt`
: (*char*) The character which indicates an end-of-line on the remote host; abbreviated `pr`; default value is `\n`. This value is used to synchronize during data transfers. The count of lines transferred during a file transfer command is based on receipt of this character.

`raise`
: (*bool*) Uppercase mapping mode; abbreviated `ra`; default value is `false`. When this mode is enabled, all lowercase letters will be mapped to uppercase by `tip` for transmission to the remote machine.

`raisechar`
: (*char*) The input character used to toggle uppercase mapping mode; abbreviated `rc`; default value is CONTROL-`@`.

`record`
: (*str*) The name of the file in which a session script is recorded; abbreviated `rec`; default value is `tip.record`.

`script`
: (*bool*) Session scripting mode; abbreviated `sc`; default is `false`. When `script` is `true`, `tip` will record everything transmitted by the remote machine in the script record file specified in `record`. If the `beautify` switch is on, only printable ASCII characters will be included in the script file (those characters between 040 and 0177). The variable `exceptions` is used to indicate characters which are an exception to the normal beautification rules.

`tabexpand`
: (*bool*) Expand tabs to spaces during file transfers; abbreviated `tab`; default value is `false`. Each tab is expanded to 8 spaces.

`verbose`
: (*bool*) Verbose mode; abbreviated `verb`; default is `true`. When verbose mode is enabled, `tip` prints messages while dialing, shows the current number of lines transferred during a file transfer operation, and more.

`SHELL`
: (*str*) The name of the shell to use for the `~!` command; default value is `/bin/sh` or taken from the environment.

HOME
: (*str*) The home directory to use for the ~c command; default value is taken from the environment.

LIMITATIONS

The full set of variables is undocumented and probably should be pared down.

FILES

/usr/ucb/tip
: Executable file

/etc/dialup
: File containing modem escape sequences

/etc/remote
: File containing global system descriptions

/etc/phones
: File containing global telephone number data base

${REMOTE}
: File containing private system descriptions

${PHONES}
: File containing private telephone numbers

~/.tiprc
: Initialization file

/usr/spool/uucp/LCK*
: Lock file to avoid conflicts with uucp

SEE ALSO

cu(1C), ftp(1N), telnet(1C), uucp(1C)

dialup(4), remote(4), phones(4) in *A/UX Programmer's Reference*

''Using cu and tip to Connect to a Remote System'' in *A/UX Networking Essentials*

NAME

touch — updates access and modification times of a file

SYNOPSIS

touch [-a] [-c] [-m] [*mmddhhmm*[*yy*]] *file...*

ARGUMENTS

-a Updates only the access times. The default is -am.

-c Silently prevents touch from creating the file if it did not previously exist.

file Specifies the file to be updated.

-m Updates only the modification times. The default is -am.

mmddhhmm
Specifies the month (mm), day (dd), hour (hh), and minute (mm) that the specified *file* is updated.

yy Specifies the year that the specified *file* is updated.

DESCRIPTION

touch causes the access and modification times of each argument to be updated. The filename is created if it does not exist. If no time is specified (see date(1)) the current time is used.

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).

EXAMPLES

Enter this command:

```
touch filea fileb
```

to set the ''date last modified'' of the two files to the current date.

LIMITATIONS

You can't touch a numeric filename without preceding that filename with the date or with a non-numeric filename on the command line. For example,

```
touch 100
```

will not work, however

```
touch 0723093584 100
```

or

```
touch file1 100
```

will work.

FILES

`/bin/touch`
    Executable file

SEE ALSO

`date`(1)

`utime`(2) in *A/UX Programmer's Reference*

NAME

`tp` — copies files to or from a `tp` archive

SYNOPSIS

`tp d`[[0...7] [`i`] [`m`] [`v`] [`w`]] [*file-in-archive*]...

`tp r`[[0...7] [`c`] [`i`] [`m`] [`v`] [`w`]] [*file-in-archive*]...

`tp t`[[0...7] [`i`] [`m`] [`v`] [`w`]] [*file-in-archive*]...

`tp u`[[0...7] [`c`] [`i`] [`m`] [`v`] [`w`]] [*file-in-archive*]...

`tp x`[[0...7] [`f`] [`i`] [`m`] [`v`] [`w`]] [*file-in-archive*]...

ARGUMENTS

`0...7`
: Selects the drive on which the tape is mounted. Normally, `0` is the default.

`-c` Creates a fresh dump; the tape directory is cleared before beginning. This option is assumed with magnetic tape since it is impossible to selectively overwrite magnetic tape.

`-d` Deletes the named *file-in-archive* from the tape. At least one name argument must be given. This function is not permitted on magnetic tapes.

`-f` Uses the first *file-in-archive* rather than a tape, as the archive. This option is known to work only with the `x` function.

file-in-archive
: Specifies a file that resides in the archive from which data is being read. If *file-in-archive* is a directory, the program refers to the files and (recursively) subdirectories of that directory.

`-i` Notes the errors when reading and writing the tape, but no action is taken. Normally, errors cause a return to the command level.

`-m` Specifies magnetic tape as opposed to DECtape.

`-r` Writes a *file-in-archive* onto the tape. If files with the same names already exist, they are replaced. ''Same'' is determined by string comparison, so `.abc` can never be the same as `/usr/sbo/abc` even if `/usr/sbo` is the current directory. If *file-in-archive* is not given, `.` is the default.

`-t` Lists the names of the specified *file-in-archive.* If *file-in-archive* is not given, the entire contents of the tape is listed.

`-u` Updates the tape. `u` is like `r`, but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. `u` is the default command if none is given.

`-v` Types the name of each file it treats preceded by the function letter. Normally `tp` does its work silently. When used with the `t` function, `v` gives more information about the tape entries than just the name.

`-w` Pauses before treating each file, types the indicative letter and the filename (as with `v`) and awaits the user's response. Response `y` means ''yes'', so the file is treated. Null response means ''no'', and the file does not take part in whatever is being done. Response `x` means ''exit''; the `tp` command terminates immediately. When used with the `x` function, files previously asked about have been extracted already. When used with the `r`, `u`, and `d` functions, no change has been made to the tape.

`-x` Extracts *file-in-archive* from the tape to the file system. The owner and mode are restored. If *file-in-archive* is not given, the entire contents of the tape are extracted.

DESCRIPTION

`tp` saves and restores files within an archive, which frequently takes the form of magnetic tape media. Its actions are controlled by the *key* argument.

The `tp` program is useful for importing tapes made on older systems.

EXAMPLES

Enter this command:

```
tp x file1
```

to extract `file1` from a `tp` formatted magnetic tape mounted on drive 0.

STATUS MESSAGES AND VALUES

Several; the nonobvious one is `Phase error`, which means the file changed after it was selected for dumping but before it was dumped.

LIMITATIONS

A single file with several links to it is treated like several files.

Binary-coded control information makes magnetic tapes written by `tp` difficult to carry to other machines; `tar` avoids the problem.

`tp` does not copy zero-length files to tape.

FILES

`/bin/tp`
: Executable file

`/dev/tap?`
: Device file

`/dev/mt?`
: Device file

SEE ALSO

`ar`(1), `cpio`(1), `tar`(1)

`dump.bsd`(1M) in *A/UX System Administrator's Reference*

NAME

`tplot` — interprets plotter instructions for use at a vintage display device

SYNOPSIS

`tplot` [`-T`*terminal* [`-e` *raster-file*]]

ARGUMENTS

`-e` *raster-file*

Causes a previously scan-converted file, *raster-file*, to be sent to the plotter.

`-T`*terminal*

Specifies a terminal that is to be given the plotting instructions. If a *terminal* is not specified, the environment parameter `$TERM` (see `environ`(5)) is used.

DESCRIPTION

These commands read plotting instructions (see `plot`(4)) from the standard input and in general produce, on the standard output, plotting instructions suitable for a particular *terminal*. 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.

EXAMPLES

The command:

```
tplot -T4014 graph.out
```

will use the encoded information in `graph.out` to plot a graph on a Tektronix 4014-type terminal.

FILES

`/bin/tplot`

Executable file

`/usr/lib/t300`

Plotting instructions file

`/usr/lib/t300s`

Plotting instructions file

`/usr/lib/t450`
: Plotting instructions file

`/usr/lib/t4014`
: Plotting instructions file

`/usr/lib/vplot`
: Plotting instructions file

`/usr/tmp/raster$$`
: Temporary file

SEE ALSO

`plot`(3X), `plot`(4), `term`(4) in *A/UX Programmer's Reference*

NAME

tput — queries terminfo database

SYNOPSIS

tput [-T*type*] *capname*

ARGUMENTS

-T*type*	Indicates the type of terminal. Normally this option is unnecessary, as the default is taken from the environment variable $TERM.
capname	Indicates the attribute from the terminfo database. See terminfo(4).

DESCRIPTION

tput uses the terminfo database to make terminal-dependent capabilities and information available to the shell. The tput command generates a string if the attribute (capability name) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, tput simply sets the exit code (0 for TRUE, 1 for FALSE), and generates no output.

EXAMPLES

To echo the clear-screen sequence for the current terminal, enter:

```
tput clear
```

To print the number of columns for the current terminal, enter:

```
tput cols
```

To print the number of columns for the 450 terminal, enter:

```
tput -T450 cols
```

To set the shell variable bold to standout mode sequence for current terminal, enter:

```
bold='tput smso'
```

This might be followed by a prompt:

```
echo "${bold}Please type in your name: \c"
```

To set exit code to indicate if current terminal is a hardcopy terminal, enter:

```
tput hc
```

STATUS MESSAGES AND VALUES

-1	Usage error
-2	Bad terminal type
-3	Bad *capname*

If a *capname* is requested for a terminal that has no value for that *capname* (for example, `tput -T450 lines`), `-1` is printed.

FILES

`/usr/bin/tput`
    Executable file

`/usr/lib/terminfo/?/*`
    Terminal descriptor files

`/usr/include/term.h`
    Definition files

`/usr/include/curses.h`
    Terminal information file

SEE ALSO

`stty`(1)

`terminfo`(4) in *A/UX Programmer's Reference*

NAME

tr — translates characters

SYNOPSIS

tr [-c] [-d] [-s] [*string1* [*string2*]]

ARGUMENTS

-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 in *string2* into single characters.

string1 [*string2*]
Specifies the input characters (*string1*) and the output characters (*string2*).

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into *string1* and *string2*:

[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.

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. For the substitution to work correctly, *string2* must have at least as many characters as *string1*; excess characters in either string are ignored by tr.

Similarly, when using the -c option, *string1* must have at least as many characters as the complement of *string1*.

EXAMPLES

To create a list of all the words in file1, one per line in file2, enter:

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >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 newline.) This was accomplished via the following translations: `tr` substitutes the newline character for all the alphabetics in `file1`, reconstitutes the alphabetics with the `-c` option, squeezes the newlines to one per occurrence with the `-s` option, and directs the output to `file2`.

LIMITATIONS

Won't handle ASCII `NUL` in *string1* or *string2*; always deletes `NUL` from input.

FILES

`/usr/bin/tr`
Executable file

SEE ALSO

`dd`(1), `ed`(1), `sh`(1)

`ascii`(5) in *A/UX Programmer's Reference*

''Other Text Processing Tools'' in *A/UX Text Processing Tools*

NAME

`troff` — formats and typesets files

SYNOPSIS

`troff` [`-`] [`-a`] [`-i`] [`-m`*name*] [`-n`*N*] [`-o`*list*] [`-q`] [`-r`*aN*] [`-s`*N*] [`-T`*dest*] [*file*...]

ARGUMENTS

`-` Specifies a filename corresponding to the standard input.

`-a` Sends a printable ASCII approximation of the results to the standard output.

file Specifies the file to be processed through `troff`. If this argument is not present, the standard input is read.

`-i` Reads standard input after the input files are exhausted.

`-m`*name*
Inserts the `/usr/lib/tmac/tmac.`*name* macro file at the beginning of the input *files.*

`-n`*N*
Numbers the first generated page *N*.

`-o`*list*
Prints only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N*-*M* means pages *N* through *M*; an initial -*N* means from the beginning to page *N*; and a final *N*- means from *N* to the end. (See LIMITATIONS, later in this section.)

`-q` Invokes the simultaneous input-output mode of the `.rd` request.

`-r`*aN*
Sets register *a* (one character name) to *N*.

`-s`*N*
Generates output to encourage typesetter to stop every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed.

`-T`*dest*
Prepares output for device *dest*, which may be a laser printer or a typesetter. For POSTSCRIPT output destined for an Apple LaserWriter, use `-Tpsc`, and pipe the output to the POSTSCRIPT filter `psdit`.

The supported typesetter is the Autologic APS-5 (`-Taps`). For output destined for an Apple ImageWriter II printer, use the `-Tiw` option and pipe the output to `daiw`(1). Other output devices may be available.

DESCRIPTION

`troff` formats text in the named *files* for printing on a phototypesetter. It is the new ''device-independent'' version of `troff`.

EXAMPLES

To request the formatting of pages 4, 8, 9, and 10 of a document contained in the files named *file1* and *file2*, and to invoke the `abc` macro package, enter:

```
troff -o4,8-10 -mabc file1 file2
```

LIMITATIONS

The `.tl` request may not be used before the first break-producing request in the input to `troff`.

`troff` recognizes only Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that `troff` generates may be off by one day from your idea of what the date is.

When `troff` is used with the `-o`*list* option inside a pipeline (for example, with one or more of `cw`, `eqn`, and `tbl`), it may cause a harmless `broken pipe` message if the last page of the document is not specified in *list*.

FILES

`/bin/troff`
: Executable file

`/usr/lib/suftab`
: Suffix hyphenation tables file

`/tmp/ta$#`
: Temporary file

`/tmp/trtmp*`
: Temporary file

`/usr/lib/tmac/tmac.*`
: Standard macro files and pointers file

`/usr/lib/macros/*`
: Standard macro files

`/usr/lib/font/dev*/*`
: Font width tables files

SEE ALSO

`checknr`(1), `cw`(1), `daps`(1), `daiw`(1), `deroff`(1), `eqn`(1), `grap`(1), `mmt`(1), `nroff`(1), `otroff`(1), `pic`(1), `psdit`(1), `tbl`(1), `tc`(1)

`mm`(5), `ms`(5), `mv`(5) in *A/UX Programmer's Reference*

''`nroff/troff` Reference'' in *A/UX Text Processing Tools*

``Introduction to `troff` and `mm`'' in *A/UX Text Processing Tools*

NAME

`true`, `false` — provides truth values

SYNOPSIS

`true`

`false`

DESCRIPTION

`true` does nothing, returning an exit status of zero.

`false` does nothing, returning a nonzero exit status.

They are typically used in input to `sh` and/or `ksh`.

EXAMPLES

```
while true
do
        command
done
```

STATUS MESSAGES AND VALUES

The `true` command has exit status zero, `false` has exit status nonzero.

FILES

`/bin/true`
: Executable file

`/bin/false`
: Executable file

SEE ALSO

`ksh`(1), `sh`(1)

NAME

`tset`, `reset` — set or reset the terminal to a sensible state

SYNOPSIS

`tset` [`-`] [`-a` *type*] [`-A`] [`-d` *type*] [`-e`*c*] [`-E`*c*] [`-k`*c*] [`-l`] [`-m` *port*] [`-p` *type*] [`-Q`] [`-r`] [`-s`] [`-S`]

`reset`

ARGUMENTS

`-` Prints the terminal type on the standard output. For compatibility with earlier versions of `tset`, this option is accepted, but its use is discouraged.

`-a` *type*
: Operates the same as the `-m arpanet:`*type*. For compatibility with earlier versions of `tset`, this option is accepted, but its use is discouraged.

`-A` Prompts the user for the terminal type.

`-d` *type*
: Operates the same as the `-m dialup:`*type*. For compatibility with earlier versions of `tset`, this option is accepted, but its use is discouraged.

`-e`*c* Sets the erase character to be the named character *c* on all terminals, the default being the backspace character on the terminal, usually CONTROL-H.

`-E`*c* Sets the erase character to *c* only if the terminal can backspace. For compatibility with earlier versions of `tset`, this option is accepted, but its use is discouraged.

`-k`*c* Sets the line kill character to be the named character *c* on all terminals. The *c* character defaults to CONTROL-X (for purely historical reasons); CONTROL-U is the preferred setting. No kill processing is done if `-k` is not specified.

`-l` Suppresses outputting terminal initialization strings.

`-m` *port*
: Specifies which terminal type is used on the given port type identifier, *port*, an optional baud specification. and the terminal type to be used if the mapping conditions are satisfied. If more than one mapping is specified, the first applicable mapping prevails. A missing type identifier matches all identifiers.

To avoid problems with metacharacters, it is best to place the entire argument to the `-m` (map) option within quotation marks (`' '`). Users of `csh` must also put a \ before any `!` used here.

`-p` *type*
: Operates the same as the `-m plugboard:`*type*. For compatibility with earlier versions of `tset`, this option is accepted, but its use is discouraged.

`-Q` Suppresses printing the `Erase set to` and `Kill set to` messages.

`-r` Prints the terminal type on the diagnostic output. For compatibility with earlier versions of `tset`, this option is accepted, but its use is discouraged.

`-s` Returns the terminal type as specified by the `-m` option, and information about the terminal to a shell's environment. When using the Bourne shell, `sh`, the command:

```
eval 'tset -s option ... '
```

or when using the C shell, `csh`, the command:

```
tset -s options... > tset$$
source tset$$
rm tset$$
```

generates as output a sequence of shell commands which place the variables `TERM` and `TERMCAP` in the environment; see `environ`(5).

`-S` Outputs the strings to be assigned to `TERM` and `TERMCAP` in the environment, rather than commands for a shell.

DESCRIPTION

`tset` causes terminal-dependent processing, such as setting erase and kill characters, setting or resetting delays, and so on. It first determines the *type* of terminal involved, names for which are specified by the `/etc/termcap` data base, and then does necessary initializations and mode settings. In the case where no argument types are specified, `tset` simply reads the terminal type out of the environment variable `TERM` and reinitializes the terminal. The rest of this discussion concerns itself with type initialization, typically done once at login, and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script `.profile` (for `sh` users) or `.login` (for `csh` users), it is desirable to give information about the types of terminals usually used when connecting to the computer through a modem. These ports are initially identified as being `dialup`, `plugboard`, or `arpanet`, and so on.

Bauds are specified as with `stty`, and are compared with the speed of the diagnostic output (which is almost always the control terminal). The baud test may be any combination of: `>`, `=`, `<`, `@`, and `!`; `@` is a synonym for `=` and `!` inverts the sense of the test.

Once it knows the terminal type, `tset` engages in terminal mode setting. This normally involves sending an initialization sequence to the terminal and setting the single character erase (and optionally the line-kill (full line erase)) characters. `tset` reports these settings by printing the diagnostic messages `Kill set to` *c* and `Erase set to` *c* on the standard error output, unless the `-Q` option is specified.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character (`#` on standard systems), the erase character is changed to a CONTROL-H (`backspace`).

`reset` sets the terminal to cooked mode, turns off `cbreak` and `raw` modes, turns on `nl`, and restores special undefined characters to their default values.

This is most useful after a program dies, leaving a terminal in a funny state; you have to type newline `reset` newline to get it to work, since RETURN (CONTROL-M) may not be recognized in this state; often none of the input will be echoed.

It is a good idea to follow `reset` with `tset`.

EXAMPLES

If the port in use is a dialup at a speed greater than 300 baud or if the port is a dialup at a speed of 300 baud or less, you can set the terminal type to an `adm3a` or to a `dw2`, respectively, by entering:

```
tset -m 'dialup>300:adm3a' -m dialup:dw2 \
-m 'plugboard:?adm3a'
```

Note: The above command can be entered on one line by omitting the backslash character.

If the *type* argument begins with a question mark (such as `?adm3a`), you are asked if you really want that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. Therefore, in this case, you will be queried on a plugboard port as to whether you are using an `adm3a`. For other ports, the port type will be taken from the `/etc/ttytype` file or a final, the default *type* option may be given on the command line, not preceded by a `-m` option.

A typical `.login` file for a `csh` user that invokes `tset` would be:

```
set noglob
set term = ('tset -e -S -r -d\?h19')
```

```
   setenv TERM "$term[1]"
   setenv TERMCAP "$term[2]"
   unset term noglob
```

This `.login` sets the environment variables `TERM` and `TERMCAP` for the user's current terminal according to the file `/etc/ttytype`. If the terminal line is a dialup line, the user is prompted for the proper terminal type.

```
   reset
```

returns the user's terminal to a usable state after being accidentally set by an interrupted process.

LIMITATIONS

Should be merged with `stty`.

`reset` doesn't set tabs properly; it can't intuitively read personal choices for interrupt and line kill characters, so it leaves these set to the local system standards.

It could be well argued that the shell should be responsible for insuring that the terminal remains in a sane state; this would eliminate the need for this program.

FILES

`/bin/tset`
`/bin/reset`
    Executable file
`/etc/ttytype`
    File containing terminal types
`/etc/termcap`
    Terminal capabilities file

SEE ALSO

`csh`(1), `sh`(1), `stty`(1)

`termcap`(4), `ttytype`(4), `environ`(5) in *A/UX Programmer's Reference*

NAME

`tsort` — sorts lines in a file topologically

SYNOPSIS

`tsort` [*file*]

ARGUMENTS

file Specifies the input file on which the sort will be performed. If this option is not specified, the standard input is understood.

DESCRIPTION

`tsort` produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

EXAMPLES

To build a new library from existing `.o` files, enter:

```
ar cr library 'lorder *.o | tsort'
```

STATUS MESSAGES AND VALUES

`Odd data`: there is an odd number of fields in the input file.

LIMITATIONS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

FILES

`/usr/bin/tsort`
Executable file

SEE ALSO

`lorder`(1), `sort`(1), `sortbib`(1)

NAME

`tty` — obtains the device filename for the terminal or CommandShell window where it is invoked

SYNOPSIS

`tty [-l] [-s]`

ARGUMENTS

`-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's pathname, allowing one to test just the exit code.

DESCRIPTION

`tty` displays the pathname of the terminal currently being used.

EXAMPLES

If you are using tty7 and you enter:

```
tty
```

the following will display:

```
/dev/tty7
```

STATUS MESSAGES AND VALUES

`2` if invalid options were specified

`0` if standard input is a terminal

`1` otherwise

`not on an active synchronous line`
if the standard input is not a synchronous terminal and the `-l` option is specified.

`not a tty`
if the standard input is not a terminal and the `-s` option is not specified.

FILES

`/bin/tty`
Executable file

SEE ALSO

`ttyname`(3C) in *A/UX Programmer's Reference*

See machid(1)

See machid(1)

See machid(1)

See machid(1)

NAME

`ucbdiff` — reports differences between two files or directories

SYNOPSIS

`ucbdiff` [`-b`] [`-c`] [`-e`] [`-f`] [`-h`] [`-i`] [`-l`] [`-n`] [`-r`] [`-s`] [`-S` *file*] [`-t`] [`-w`] *dir1 dir2*

`ucbdiff` [`-b`] [`-c`] [`-e`] [`-f`] [`-h`] [`-i`] [`-n`] [`-t`] [`-w`] *file1 file2*

`ucbdiff` [`-b`] [`-D`*string*] [`-i`] [`-w`] *file1 file2*

ARGUMENTS

`-b` Ignores trailing blanks (spaces and tabs) and other strings of blanks to compare equal.

`-c` Produces a `ucbdiff` with lines of context. The default is to present 3 lines of context; this may be changed, for example to 10, by `-c10`. With `-c`, the output format is modified slightly. The output begins with an identification of the files involved and their creation dates; then each change is separated by a line with a dozen `*`'s. The lines removed from *file1* are marked with a `-`; those added to *file2* are marked `+`. Lines which are changed from one file to the other are marked in both files with a `!`.

Changes that are separated by fewer than the number of lines in the current context are grouped together on output. (This is a change from the previous `ucbdiff -c` but the resulting output is usually much easier to interpret.)

`-e` Produces a script of `a`, `c`, and `d` commands for the `ed`, editor which will recreate *file2* from *file1*. 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 `ucbdiff` need be on hand. A "latest version" appears on the standard output.

```
(shift;cat $*;echo '1,$p')|ed - $1
```

Extra commands are added to the output when comparing directories with `-e`, so that the result is a `sh` script for converting text files common to the two directories from their state in *dir1* to their state in *dir2*.

`-f` Produces a script similar to that of `-e`, but in the opposite order and not useful with `ed`.

`-h` Does a fast, half-hearted job. It works only when changed stretches are short and well separated, but will work on files of unlimited length.

`-i` Ignores the case of letters. For example, `A` will compare equal to `a`.

`-n` Produces a script similar to that of `-e`, but in the opposite order and with a count of changed lines on each insert or delete command. This is the form used by `rcsdiff`.

`-t` Expands tabs in output lines. Normal or `-c` output adds character(s) to the front of each line, which may alter the indentation of the original source lines and make the output listing difficult to interpret. This option will preserve the original source's indentation.

`-w` Totally ignores whitespace (blanks and tabs). This option is similar to `-b`. For example, the following two lines will compare equal.

```
if ( a == b )
if(a==b)
```

dir1 dir2
Specifies the two directories to be compared.

`-D`*string*
Creates a merged version of *file1* and *file2* on the standard output with C preprocessor controls included, so that a compilation of the result without defining *string* is equivalent to compiling *file1*, while defining *string* will yield *file2*.

file1 file2
Specifies the two files to be compared.

`-l` Displays long output format. Each set of text file differences is piped through `pr` to paginate the output; other differences are remembered and summarized after all text file differences are reported.

`-r` Causes application of `ucbdiff` recursively to common subdirectories encountered.

`-s` Reports files which are the same and are otherwise not mentioned.

`-S`*file*
Starts a directory `ucbdiff` in the middle, beginning with specified *file*.

DESCRIPTION

`ucbdiff` is used by the `rcs(1)` Revision Control System. If both arguments are directories, `ucbdiff` sorts the contents of the directories by name and then runs the regular file `diff` algorithm (described later in this section) on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed.

When `ucbdiff` is run on regular files and when comparing text files which differ during directory comparison, `ucbdiff` tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, `ucbdiff` finds the smallest sufficient set of file differences. If neither *file1* nor *file2* is a directory, then either may be given as `-`, in which case the standard input is used. If *file1* is a directory, then a file in that directory whose filename is the same as the filename of *file2* is used (and vice versa).

There are several options for output format; the default output format 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 how to equally 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 are all the lines affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

STATUS MESSAGES AND VALUES

Exit status is 0 for no differences, 1 for some, 2 for trouble.

LIMITATIONS

Editing scripts produced under the `-e` or `-f` option are naive about creating lines consisting of a single period (.).

When comparing directories with the `-b`, `-w`, or `-i` options specified, `ucbdiff` first compares the files like `cmp` and then decides to run the `ucbdiff` algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string or case differences.

FILES

`/usr/ucb/ucbdiff`
    Executable file
`/tmp/d`*?????*
    Temporary file
`/usr/lib/ucbdiffh`
    File used with the `-h` option
`/bin/diff`
    Executable file used for directory comparisons

`/bin/pr`
: Executable file

SEE ALSO

`cc`(1), `cmp`(1), `comm`(1), `diff`(1), `ed`(1), `rcs`(1), `ucbdiff3`(1)

NAME

`ucbdiff3` — reports the differences between three files

SYNOPSIS

`ucbdiff3` [`-e`] [`-E`] [`-x`[*-3*]] [`-X`[*-3*]] *ver1 ver2 ver3*

ARGUMENTS

`-e` Publishes a script for the `ed` editor that will incorporate into *ver1* all changes between *ver2* and *ver3* (the changes that would normally be flagged `====` and `====3`).

`-E` Publishes a script for the `ed` editor that will incorporate into *ver1* all changes between *ver2* and *ver3*, but treat overlapping changes (changes that would be flagged with `====` in the normal listing) differently. The overlapping lines from both files will be inserted by the edit script, bracketed by <<<<<< and >>>>>> lines.

ver1 ver2 ver3
Specifies three versions of a file.

`-x`[*-3*]
Produces a script to incorporate only the changes flagged `====` (`====3`).

`-X`[*-3*]
Produces a script to incorporate only the changes that would be flagged `====` (`====3`), in the normal listing differently. The changes from both files will be inserted by the edit script, bracketed by <<<<<< and >>>>>> lines.

DESCRIPTION

`ucbdiff3` is used by the `rcs` Revision Control System. The `ucbdiff3` command compares three versions of a file and publishes disagreeing ranges of text flagged with these codes.

`====`	all three files differ
`====1`	*ver1* is different
`====2`	*ver2* is different
`====3`	*ver3* is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of two ways:

f`:`*n1*`a`	Appends text after line number *n1* in file *f*, where *f* = 1, 2, or 3.
f`:`*n1*`,`*n2*`c`	Changes text in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a `c` indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

EXAMPLES

Suppose lines 7-8 are changed in both `ver1` and `ver2`. Applying the edit script generated by the command

```
ucbdiff3 -E ver1 ver2 ver3
```

to `ver1` results in the file:

lines 1-6 of `ver1`
`<<<<<<< ver1`
lines 7-8 of `ver1`
`=======`
lines 7-8 of `ver3`
`>>>>>>> ver3`
rest of `ver1`

The `-E` option is used by RCS `merge` to insure that overlapping changes in the merged files are preserved and brought to someone's attention.

LIMITATIONS

Text lines that consist of a single period (.) will defeat the `-e` option.

FILES

`/usr/ucb/ucbdiff3`
Executable file
`/tmp/d3?????`
Temporary file
`/usr/lib/ucbdiff3`
Library file

SEE ALSO

`diff`(1), `diff3`(1), `rcs`(1), `ucbdiff`(1)

NAME

`ul` — filters special underlining sequences imbedded in text for use at a display device

SYNOPSIS

`ul [-t` *terminal*] [*file*]...

ARGUMENTS

file Specifies the file that is to be run through the filter.

`-t` *terminal*
: Specifies the type of terminal being used. If this option is not given, the environment is searched, and if necessary, `/usr/lib/terminfo` is read to determine the appropriate sequences for underlining. If none of the fields `us`, `ue`, or `uc` are present, and if `so` and `se` are present, standout mode is used to indicate underlining.

DESCRIPTION

`ul` reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining.

If the terminal can overstrike, or handles underlining automatically, `ul` behaves like `cat`. If the terminal cannot underline, underlining is ignored.

EXAMPLES

Enter this command:

```
ul file1
```

to display `file1` on the terminal with underlined portions of the file either underlined, or in reverse video when this option is supported for the terminal.

LIMITATIONS

The `nroff` program usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

FILES

`/usr/bin/ul`
: Executable file

`/bin/cat`
: Executable file

`/usr/lib/terminfo`
: Terminal information file

SEE ALSO
colcrt(1), man(1), nroff(1)

NAME

`uname` — displays identification information about the current system

SYNOPSIS

`uname [-a] [-m] [-n] [-r] [-s] [-v]`

ARGUMENTS

`-a` Displays the system name, nodename, operating system release, operating system version, and the machine hardware name.

`-m` Displays the machine hardware name.

`-n` Displays the nodename. The nodename may be a name that the system is known by to a communications network.

`-r` Displays the operating system release.

`-s` Displays the system name. This option is the default.

`-v` Displays the operating system version.

DESCRIPTION

`uname` displays the name of the current system on the standard output file. It is mainly useful to determine which system is being used. The options cause selected information returned by `uname` to be displayed.

EXAMPLES

If you enter:

```
uuname
```

from an A/UX system, the following will display:

```
A/UX
```

FILES

`/bin/uname`
Executable file

SEE ALSO

`uname(2)` in *A/UX Programmer's Reference*

See compact(1)

See compress(1)

See compress(1)

See expand(1)

NAME

unget — undoes a previous get of an SCCS file

SYNOPSIS

`unget [-n] [-r`*SID*`] [-s]` *file...*

ARGUMENTS

file Specifies the affected SCCS file. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

`-n` Retains the retrieved file which would normally be removed from the current directory.

`-r`*SID*
Uniquely identifies which delta is no longer intended. (This would have been specified by `get` as the ''new delta'') The use of this option is necessary only if two or more outstanding `gets` for editing on the same SCCS file were done by the same person (login name). A message results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

`-s` Suppresses the printout, on the standard output, of the intended delta's *SID*.

DESCRIPTION

`unget` undoes the effect of a `get -e` command done prior to creating the intended new delta. If a directory is named, `unget` behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored.

The options apply independently to each named file.

EXAMPLES

The command:

```
% unget s.test1.c
1.2
```

undoes version 1.2 of `test1.c` set up for editing by an earlier `get -e` command.

STATUS MESSAGES AND VALUES

Use `help` for explanations.

FILES

`/usr/bin/unget`
Executable file

SEE ALSO

admin(1), cdc(1), comb(1), delta(1), get(1), help(1), prs(1), rmdel(1), sact(1), sccs(1), sccsdiff(1), val(1), what(1)

sccsfile(4) in *A/UX Programmer's Reference*

''SCCS Reference'' in *A/UX Programming Languages and Tools, Volume 2*

NAME

`uniq` — reports repeated lines in a file

SYNOPSIS

`uniq` [`-c`] [`-d`] [+*num*] [-*num*] [`-u`] [*infile* [*outfile*]]

ARGUMENTS

`-c` Supersedes the `-u` and `-d` option and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

`-d` Writes one copy of just the repeated lines. The normal mode output is the union of the `-u` and `-d` mode outputs.

infile
: Specifies the input file that `uniq` is going to read.

+*num*
: The first *num* characters are ignored. Fields are skipped before characters.

-*num*
: The first *num* fields together with any blanks before each are ignored. A field is defined as a string of nonspace, nontab characters separated by tabs and spaces from its neighbors.

outfile
: Specifies the output file into which the repeated lines will be written.

`-u` Outputs only the lines that are not repeated in the original file.

DESCRIPTION

`uniq` reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written in the output file. *infile* and *outfile* should always be different. Note that repeated lines must be adjacent in order to be found; see `sort`(1).

EXAMPLES

To print the contents of `file1` with adjacent identical lines removed, enter:

```
uniq file1
```

FILES

`/usr/bin/uniq`
: Executable file

SEE ALSO

comm(1), diff(1), sort(1)

NAME

`units` — rescales quantities according to a the unit of measure specified

SYNOPSIS

`units`

DESCRIPTION

`units` converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively (see the examples).

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 (see the second example).

`units` only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

`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

The `pound` unit is not recognized as a unit of mass, the `lb` abbreviation is. Compound names are run together, (for example, `lightyear`). British units that differ from their U.S. counterparts are prefixed with `br`, thus: `brgallon`. For a complete list of units, type

```
cat /usr/lib/unittab
```

EXAMPLES

You have: `inch`
You want:

```
cm
* 2.540000e+00
/ 3.937008e-01
```

You have: `15 lbs force/in2`
You want:

```
atm
```

```
        * 1.020689e+00
        / 9.797299e-01
```

FILES

`/usr/bin/units`
: Executable file

`/usr/lib/unittab`
: Table file

See pack(1)

NAME

updater — updates files between two machines

SYNOPSIS

updater [d] [r] [u] *local remote...*

updater [p] [r] [u] *local remote...*

updater [t] [r] [u] *local remote...*

ARGUMENTS

-d Lists the difference between the files on the local and remote machines.

local
Specifies the local directory name.

-p Puts files from the local machine onto the remote machine to update the remote machine.

-r Replaces a file if it did not exist on the destination machine.

remote
Specifies the remote directory names. Only one remote name can be specified if the p (put) key is specified.

-t Takes files from the remote machine to update the local machine.

-u Updates a file only if it exists on both machines. This is the default condition.

DESCRIPTION

updater updates files between two machines.

Algorithm

Open /dev/tty0 to the remote machine.

stty the local port and send a stty command to the remote machine to condition both ends of the connection.

Send a

remote ; *sumdir* . |sort+2>

to remote machine for each remote system;

local; *sumdir* . |sort>/tmp/l"

for local machine.

Wait for remote to complete.

Take /tmp/r*XXXXX*.

Do a comparison between the local and the union of the remotes:

exists on remote only:
: If both the `t` and `r` keys are specified, take the file; otherwise list the file.

exists on local only:
: If both `p` and `r` keys are specified, put the file; otherwise list the file.

exist on both but different:
: If `t` key is specified, take the file.
If `p` key is specified, put the file.
If `d` key is specified, list the file.

same:
: nothing

EXAMPLES

To use `/dev/tty0` to communicate with a remote machine and compare directories on the remote and local systems, enter:

```
updater d ...
```

FILES

`/usr/bin/updater`
: Executable file

NAME

`uptime` — reports how long system has been up

SYNOPSIS

`uptime`

DESCRIPTION

`uptime` prints the current time, the length of time the system has been up, the number of users currently logged into the system, and the average number of jobs in the run queue over the last 1, 5, and 15 minutes. It is, essentially, the first line of a `w` command.

FILES

`/usr/ucb/uptime`
  Executable file
`/dev/kmem`
  Temporary file
`/etc/utmp`
  Temporary file

SEE ALSO

`ps`(1), `ruptime`(1N), `w`(1)

NAME

`users` — reports a list of the users who are logged on to the system

SYNOPSIS

`users` [*file*]

ARGUMENTS

file Specifies the file from which `users` reads user information. The default file is `/etc/utmp`.

DESCRIPTION

`users` lists the login names of the users currently on the system in a compact, one-line format.

FILES

`/usr/ucb/users`
Executable file

`/etc/utmp`
Temporary file

SEE ALSO

`finger`(1), `w`(1), `who`(1)

NAME

`uucp` — copies files from one system to another system

SYNOPSIS

`uucp` [`-c`] [`-C`] [`-d`] [`-f`] [`-g`*grade*] [`-j`] [`-m`] [`-n`*login-name*] [`-r`] [`-s`*file*] [`-x`*debug-level*] *source-file destination-file*

ARGUMENTS

`-c` Reads the actual source file, rather than a copy, when the file is transferred. This option is the default.

`-C` Makes a copy of the source file in `/usr/spool/uucp/`*system*`/`*grade*, where *system* is the name of the system to which the file is being sent and *grade* is the priority assigned to the transfer. If you use this option, the copy, rather than the original file, is used to make the transfer. This option is useful if you want to modify the file after submitting your `uucp` request, but do not want the modified file to be transferred.

`-d` Makes any intermediate directories at the destination. This option is the default.

destination-file
Specifies the pathname and filename of the copied file on the targeted system.

`-f` Prevents the making of any intermediate directories that may be necessary for the copy to complete successfully.

`-g`*grade*
Specifies a grade that is used to prioritize file transfers. The value of *grade* is a string of one or more alphanumeric characters. To see the available priorities, use the `uuglist` command, which displays a list of priorities or a message that says to use a single letter or number.

`-j` Writes the job identification string on the standard output. The string can be used by `uustat` to obtain the status or terminate a `uucp` job and is valid as long as the job remains queued on the local system.

`-m` Sends mail to the requester when the transfer is complete. This option does not work when you are receiving multiple files as the result of one `uucp` request. The `-s` option overrides this option.

`-n`*login-name*
Sends mail to *login-name* on the remote system to notify a user that a file has been transferred.

`-r` Queues the job, but does not begin the transfer.

`-s`*file*
Writes status information about the transfer into *file*. This option

overrides the `-m` option.

source-file

Specifies the source and destination of a file that `uucp` is to copy. Both arguments have this form:

[[*system* !]...]*pathname*

The *system* portion of the argument must be a system that is known to `uucp` on your system. You can use the `uuname` command to learn the names of systems that are known on your system. See `uucico`(1M) for additional information. See *A/UX Network System Administration* for details.

The *pathname* portion of source and destination arguments can contain the shell metacharacters `?`, `*`, `~`, and `[ ]`, which are expanded on the appropriate system. In most cases, shell metacharacters must be protected from expansion by the local shell by enclosing the command in double quotation marks (`"`), or by preceding the metacharacter with a backslash (`\`). You specify the *pathname* portion of the source and destination arguments by using one of the following forms:

- An absolute or relative pathname.
- An optional pathname preceded by ~*login-name* where *login-name* is a login name on the destination system; the value of ~*login-name* is expanded to that user's home directory.
- A pathname of ~/*file*. In this case, `uucp` forms the pathname by appending ~/*file* to the string `/usr/spool/uucppublic`. The *file* portion of the pathname is treated as a filename if just one file is being transferred by this request. To override the treatment of *file* as a file, follow it with `/`. For example, `~/dan/` causes the `/usr/spool/uucppublic/dan` directory to be made if it does not exist and to be the repository of the transferred file. If the *source-file* argument expands to multiple files or if *file* is already a directory, *file* is treated as a directory.
- The current directory.

If the resulting pathname does not exist on the remote system, the copy operation fails.

If *destination-file* is a directory, the basename of *source-file* is used to name the transferred file.

`-x`*debug-level*

Writes debugging information on standard output. The value of *debug-level* is a number from 0 to 9. Higher numbers produce more detailed debugging information.

DESCRIPTION

uucp copies files from one system to another system, using both serial and Ethernet connections. The successful completion of a uucp command depends on the correct configuration of uucp files on your system. For detailed information about configuring uucp, see *A/UX Network System Administration.*

EXAMPLES

The following command transfers the file myfile, which resides in the current directory, to the home directory of a user called art on a system named hugo:

```
uucp myfile hugo!~art
```

The success of this command depends on your permission to read the file and the permission of uucp on the system named hugo to write the file in the home directory of the user called art.

The following command transfers the files that begin with R in the /tmp directory to the /usr/spool/uucppublic directory on a system named sparkie. If there are other files queued for tweetie, these files are sent before files of a lower priority. The command also sends mail to the requester when the transfer to tweetie completes and causes mail to be sent to a user called nancy on sparkie when the files arrive on sparkie.

```
uucp -ghigh -m -nnancy /tmp/R* \
tweetie!junebug!sparkie!~
```

The success of this command depends on the ability of the systems named tweetie and junebug to forward files.

WARNINGS

Some versions of uucp cannot forward files. When specifying more than one system name in a source or destination file argument, take care to ensure that intermediate systems along the route are able to forward files. If an intermediate system cannot forward, the file transfer will fail. The A/UX Release 3.0 version of uucp forwards files correctly if the COMMANDS keyword in the Permissions file includes uucp. See *A/UX Network System Administration* for details.

For security reasons, your access to remote files and directories may be severely restricted when you are both sending and receiving files. You can ask a responsible person on the remote system to copy the files to an accessible directory, such as /usr/spool/uucppublic, or send them to you.

Regardless of the source file's owner and permission bit settings, when the transfer is complete, the transferred copy of the file is owned by `uucp` and the file's permission bits are set to 666 (read and write permission for owner, group, and others).

LIMITATIONS

If you are logged in as `root` and try to send a file in a directory that is unsearchable by other users or try to send a file that is unreadable by other users, `uucp` will fail with a `can't read` error message.

FILES

`/usr/bin/uucp`
: Executable file

`/usr/lib/uucp/*`
: Directory of `uucp` commands and configuration files

`/usr/lib/uucp/uucico`
: Executable file that actually transfers files

`/usr/spool/uucp`
: Directories containing the queued jobs

`/usr/spool/uucppublic/*`
: Public directory for receiving and sending files

SEE ALSO

`basename`(1), `mail`(1), `uuglist`(1C), `uusend`(1C), `uustat`(1C), `uux`(1C)

`uucico`(1M), `uuxqt`(1M) in *A/UX System Administrator's Reference*

Chapter 8, ''Setting Up the UUCP System,'' in *A/UX Network System Administration*

See uuencode(1C)

NAME

`uuencode`, `uudecode` — encode and decode a binary file

SYNOPSIS

`uuencode` [*source-file*] *decoded-name*

`uudecode` [*encoded-file*]

ARGUMENTS

decoded-name
: Specifies that name that the file is to have when it is decoded by `uudecode`. This argument is required. The `uuencode` command stores *decoded-name*, in the encoded file's header, as well as the permission mode of *source-file*, for use by `uudecode`.

encoded-file
: Specifies the name of a file to be decoded. If you do not specify this argument, `uudecode` reads from the standard input.

source-file
: Specifies the name of a file to be encoded. If you do not specify this argument, `uuencode` reads from the standard input.

DESCRIPTION

`uuencode` reads a binary file, such as an executable program, converts the data to ASCII representation, and writes the converted data to the standard output. The ASCII representation of the file can then be transferred by programs that handle only ASCII data, such as `mail`. The encoded file is an ordinary ASCII text file that you can edit with any text editor. But it is best only to change the permission mode and the value of *encoded-file*, stored in the header, which is the first line of the output, to avoid corrupting the decoded binary.

The `uudecode` command converts an encoded file into its normal binary representation, removes any leading and trailing lines that may be added by mail programs, and creates the file so that it has the name and permission mode that is stored in the header of *encoded-file.*

LIMITATIONS

The encoded file's size is expanded by 35 percent; 3 bytes become 4, plus control information, causing it to take longer to transmit than the equivalent binary.

When you run `uudecode`, you must have permission to write the file by using the name stored in the header of the encoded file. In addition, `uudecode` is owned by `uucp` and fails with a `permission denied` message if you run it in a directory whose permission mode does not allow other users to write.

FILES

`/usr/bin/uudecode`
: Executable file

`/usr/bin/uuencode`
: Executable file

SEE ALSO

`mail`(1), `uucp`(1C), `uux`(1C)

NAME

uuglist — displays the service grades that are available on your system

SYNOPSIS

uuglist [-l] [-u] [-x*debug-level*]

ARGUMENTS

-l Displays the correspondence of configurable service grades, as defined in the file /usr/lib/uucp/Grades, to the single-letter service grades that the UUCP system actually uses.

-u Displays the grades that you are allowed to use.

-x*debug-level*
Writes debugging information on standard output. The value of *debug-level* is a number from 0 to 9. Higher numbers produce more detailed debugging information.

DESCRIPTION

uuglist displays all the service grades that are available for use with the -g option of uucp and uux on your system.

Internally, the UUCP system uses single-letter grades ranging from A, which has the highest priority, to z, which has the lowest priority. When uucp and uux queue a job, the control files for the job are created in a directory named for its grade (for example, the directory Z in /usr/spool/uucp/*system-name*). When uucico processes jobs, it transfers jobs that have the highest priority first. The default grade is Z.

Your system administrator can use the Grades file to associate a particular word (such as high), with a letter, such as A, so that you can use words, rather than letters, as arguments to the -g option of the uucp and uux commands. To see the words that are associated with grades, use the -l option to uuglist. The Grades file can also be used to prevent users from using certain grades. To see the grades that you can use, use the -u option to uuglist.

STATUS MESSAGES AND VALUES

The uuglist command displays this message when the file /usr/lib/uucp/Grades does not exist:

```
No administrator defined service grades available
on this machine, use single letter/number only.
```

If you run uuglist when the /usr/lib/uucp/Grades file contains invalid data, one of the following messages is be mailed to the uucp user account:

```
Error encountered in the restrictions field of the
Grades file. Field contents (contents).
```

```
Error encountered in action field of the Grades file.
Field contents (contents)
```

Your system may forward these mail messages to the `root` user account.

FILES

`/usr/bin/uuglist`
: Executable file

`/usr/lib/uucp`
: Directory of `uucp` commands and configuration files

`/usr/lib/uucp/Grades`
: File that contains the available service grades

SEE ALSO

`uucp`(1C), `uux`(1C)

Appendix B, ''Customizing the UUCP System,'' in *A/UX Network System Administration*

NAME

`uulog` — displays information about `uucp` file transfers

SYNOPSIS

`uulog` [`-cqx`] [`-l`[*hours*]] [`-`*lines*] [`-f`*system*] [*system*]...

`uulog` [`-cqx`] [`-l`[*hours*]] [`-`*lines*] [`-s`*system*] [*system*]...

ARGUMENTS

`-c` Displays the contents of the `uucp` log files.

`-f`*system*
: Displays the last ten lines, and any lines that are subsequently added, to the log file for the system specified by *system.* To stop this command, send an interrupt, for example by pressing CONTROL-C.

`-l`[*hours*]
: Displays log entries that have been made within the last number of hours, as specified by *hours.* The default value of *hours* is 2.

`-`*lines*
: Specifies the number of lines to display when using the `-f`*system* option. The maximum value of *lines* is 999.

`-q` Displays the contents of the `uux` log files.

`-s`*system*
: Displays information for the system specified by *system.*

`-x` Displays the contents of the `uuxqt` log files.

DESCRIPTION

`uulog` displays the contents of the `uucico` log files. If you specify the name of a system, `uulog` displays the contents of the `uucico` log file for that system only. If you do not specify the name of a system, `uulog` displays the contents of all `uucico` log files. You can use the `-c`, `-q`, or `-x` option to display the contents of the `uucp`, `uux`, `uuxqt` log files, respectively.

FILES

`/usr/bin/uulog`
: Executable file

`/usr/lib/uucp`
: Directory of UUCP commands and configuration files

`/usr/spool/uucp/.Log/uucico/`*system*
: Log file of `uucico` transfers for *system*

`/usr/spool/uucp/.Log/uucp/`*system*
: Log files of `uucp` transfers for *system*

`/usr/spool/uucp/.Log/uux/`*system*
: Log files of `uux` transfers for *system*

`/usr/spool/uucp/.Log/uuxqt/`*system*
: Log files of `uuxqt` transfers for *system*

SEE ALSO

`uucp`(1), `uustat`(1C), `uux`(1C), `uuxqt`(1C)

Chapter 8, ''Setting Up UUCP System,'' in *A/UX Network System Administration*

NAME

`uuname` — displays the names of systems to which `uucp` and `cu` can connect

SYNOPSIS

`uuname [-c] [-l]`

ARGUMENTS

`-c` Lists the names of systems to which `cu` can automatically connect by specifying a system name. This option is useful only if your system uses the `/usr/lib/uucp/Sysfiles` file to tell `cu` to use a file other than the default `/usr/lib/uucp/Systems` file to get configuration information for remote systems. If you do not specify this option, `uuname` lists all the systems to which `uucp` can connect or, if your system does not use `/usr/lib/uucp/Sysfiles`, all the systems to which both `uucp` and `cu` can connect.

`-l` Displays the system name of the local system.

DESCRIPTION

`uuname` reads the file `/usr/lib/uucp/Systems`, and other files if specified by `/usr/lib/uucp/Sysfiles`, to get and display the names of systems to which `uucp` and `cu` have been configured to connect.

FILES

`/usr/bin/uuname`
Executable file

`/usr/lib/uucp`
Directory of `uucp` commands and configuration files

`/usr/lib/uucp/`*file*
File used exclusively by `cu` or `uucp`, as specified by the file `/usr/lib/uucp/Sysfiles`, containing configuration information for remote systems

`/usr/lib/uucp/Sysfiles`
File specifying that `cu` or `uucp` is to use a file other than the default file, `/usr/lib/uucp/Systems`, to get configuration information for remote systems

`/usr/lib/uucp/Systems`
File containing configuration information for remote systems

SEE ALSO

cu(1), uucp(1)

Chapter 8, ''Setting Up the UUCP System,'' in *A/UX Network System Administration*

See uuto(1C)

NAME

`uusend` — sends a file to a remote host

SYNOPSIS

`uusend [-m` *file-permission*`] -r` *sourcefile system1* `!`*...remotefile*

ARGUMENTS

`-m` *file-permission*
: Causes the value of *file-permission* to be used as the file's file permission when it is delivered. The value of *file-permission* is an octal number. If this argument is not specified, the file permissions of the input file are used.

`-r` Queues the job but does not start the file transfer.

remotefile
: Specifies the name of the file when it is delivered. The value of *remotefile* can be an absolute pathname or can include the ~*login-name* syntax. For example, a *remotefile* argument with the value `~mike/letter` causes the file specified by *sourcefile* to be named `letter` and placed in the home directory of the user named `mike` on the last system named in the `uusend` command line.

sourcefile
: Specifies the name of a file that is to be sent. The value of *sourcefile* can be `-`, which causes the standard input to be used.

system1 `!`*...*
: Specifies the systems involved.

DESCRIPTION

`uusend` sends a file to a given location on a remote system by acting as a front-end to the `uux` command. The system need not be directly connected to the local system, but a chain of `uucp` links must be used to connect the two systems.

STATUS MESSAGES AND VALUES

If `uucp` fails on any system beyond the first system, you will not be notified.

LIMITATIONS

All systems in a multisystem `uucp` command must have the `uusend` command available and allow remote execution of it; for that reason, `uusend` is included with this release.

Some versions of the `uux` command (but not the A/UX Release 3.0 version) prevent the input of a binary file. If this limitation exists in any system along the line, the file is severely damaged.

FILES

/usr/ucb/uusend
: Executable file

SEE ALSO

uucp(1C), uuencode(1C), uux(1C)

''Using UUCP to Connect to a Remote System" in *A/UX Networking Essentials*

NAME

`uustat` — controls `uucp` jobs and provides status information

SYNOPSIS

`uustat` [`-a`] [`-S`*job-status*] [[`-j`] [`-s`*system*]] [`-u`*login-name*] [`-x`*debug-level*]

`uustat -k`*job-id* [`-n`] [`-x`*debug-level*]

`uustat -m` [`-x`*debug-level*]

`uustat -p` [`-x`*debug-level*]

`uustat -q` [`-x`*debug-level*]

`uustat -r`*job-id* [`-n`] [`-x`*debug-level*]

`uustat -t`*system* [`-d`*minutes*] [`-c`] [`-x`*debug-level*]

ARGUMENTS

`-a` Lists all jobs in the queue. You can use this option with the `-j` option.

`-c` Reports the average queue time rather than the average transfer rate. You can use this option only in conjunction with the `-t` option.

`-d`*minutes*
Overrides the 60-minute default so that `uustat` reports statistics for the number of minutes specified by *minutes*.

`-j` Lists the total number of jobs displayed. You can use this option with the `-a` or the `-s` option.

`-k`*job-id*
Cancels the `uucp` request whose job identification is *job-id*. This option is effective only if used by the user that originated the `uucp` job or if made by a user logged in as `root` or `uucp`. If the job is canceled by a user logged in as `root` or `uucp`, `uustat` notifies the user who originated the `uucp` request by sending mail.

`-m` Reports the status of the most recent connection to all systems for which a log file exists.

`-n` Suppresses information written on the standard output, but does not suppress information written on the standard error. You can use this option with the `-k` or the `-r` option.

`-p` Causes `uustat` to examine the lock files and run the command `ps -flp` for each process ID that it finds.

`-q` Lists the jobs queued for each system. If a status file exists for the system, its date, time, and status information are displayed. If the display includes a number in parentheses next to the number of `C` or `X` files, the number in parentheses is the age in days of the oldest `C.` or

`X.` file for that system. The `Retry` column represents the number of hours until the next possible call. The `Count` column is the number of failure attempts. For systems with a moderate number of outstanding jobs, `uustat -q` may take 30 seconds or more to produce a report.

`-r`*job-id*
Rejuvenates the job ID specified by *job-id*. The modification time of the files associated with *job-id* is set to the current time, which prevents `uucleanup` from deleting the job because it has become old.

`-s`*system*
Reports the status of all `uucp` requests for the remote system specified by *system*. You can use this option in conjunction with the `-j` and `-u` options.

`-S`*job-status*
Reports the state of the job, whose type is specified by *job-status*. The value of *job-status* can be one or more of the following options in any combination:

`c`	Reports jobs that have completed successfully. The completed-state information is maintained in the accounting log, which is optional and therefore may be unavailable.
`i`	Reports interrupted jobs. A job is interrupted if the transfer began but was terminated before the file was completely transferred.
`q`	Reports queued jobs. A job is queued if the transfer has not started.
`r`	Reports running jobs. A job is running when the transfer has begun.

You can use the `-S` option with the `-a`, `-j`, `-s`, and `-u` options.

`-t`*system*
Reports the average transfer rate for the past 60 minutes for the remote system specified by *system*. These calculations are based on information contained in the optional performance log and therefore may not be available. Calculations can only be made from the time that the performance log was last cleaned up. The `-t` option produces no message when the data needed for the calculations is not being logged.

`-u`*login-name*
Reports the status of all `uucp` requests issued by the user specified by *login-name*. You can use this option in conjunction with the `-s`

option.

-x*debug-level*
: Writes debugging information on standard output. The value of *debug-level* is a number from 0 to 9. Higher numbers produce more detailed debugging information.

DESCRIPTION

uustat displays the status of UUCP jobs, cancels jobs, provides remote-system performance information in terms of average transfer rates or average queue times, and provides specific information, organized by system or by user, about connections.

The uustat command without any options displays the status and job identification number of all UUCP requests that have yet to be transferred or have recently completed successfully. You can use options to cause uustat to provide other types of information about uucp activity on your system.

You can use the -a, -j, -m, -n, -p, -q, or -r*job-id* option to get general status information. Unless otherwise noted, these options cannot appear with any other options on the uustat command line.

EXAMPLES

Here is 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 first column is the system name. The second column tells how many control files are waiting to be processed. Each control file can specify the transmisson of 0 or more files; if 0, uucico is to poll the remote system to see if it has files to send to your system. The third column reflects the date and time of the most recent attempt to connect to the system, and the fourth column represents the status of that attempt.

Here is an example that uses the -t and -c options to display the amount of time for which jobs remain in the queue for the system named eagle:

```
uustat -teagle -d50 -c
```

The command produces output in this format:

```
average queue time to eagle for last 50 minutes: 5 seconds
```

The same command without the -c option produces output in this format:

```
average transfer rate with [eagle] \
for last [50] minutes: 2000.88 bytes/sec
```

Here is an example of the output of the -s option, which displays all of the jobs queued for a specific system, and the -u option, which displays all of the jobs queued by a specific user:

```
eagleN1bd7  4/07-11:07  S  eagle  dan  522    /home/dan/A
eagleC1bd8  4/07-11:07  S  eagle  dan   59    D.3b2al2ce4924
            4/07-11:07  S  eagle  dan  rmail  mike
```

The first column is the job ID of the job. The second column is the date and time the job was queued. The third column is an `S` if the job is sending a file or an `R` if the job is requesting a file. The fourth column is the name of the system the file is being transferred to or from. The fifth column is the login name of the user who queued the job. The sixth column is the size of the file in bytes, or, in the case of a remote execution command such as `rmail`, the name of the command. When the size appears in this column, the filename is displayed in the seventh column. This filename can either be the name given by the user or an internal name, such as `D.3b2alce4924`, that is created for data files associated with remote executions.

The output of the `-S` option is the same as the output for the `-s` and `-u` options except that the job states are appended as the last output word.

Here is an example that uses the `-S` and the `c` options to display the current list of completed jobs:

```
uustat -Sc
```

Here is an example of the output of this command:

```
eagleC1bd3 completed
```

NOTES

If you use the `-C` option of `uucp`, which specifies that `uucp` is to transfer the file from the file itself rather than making a copy in `/usr/spool/uucp`, and the file is later moved or deleted before `uucp` can transfer it, `uustat` reports a file size of -99999. Such a job eventually fails because the file to be transferred cannot be found.

FILES

`/usr/lib/uucp`
: Directory of `uucp` commands and configuration files

`/usr/spool/uucp/*`
: Files, organized by system name, that contain the control information that makes up a `uucp` job

`/usr/uucp/.Admin/account`
: Log of accounting information

`/usr/uucp/.Admin/perflog`
: Log of performance information

SEE ALSO

`uucp`(1C), `uulog`(1C)

Chapter 8, ''Setting Up the UUCP System,'' in *A/UX Network System Administration*

NAME

uuto, uupick — provide an easy interface to the uucp command, using the public directories

SYNOPSIS

uuto [-m] [-p] *file... destination*

uupick [-s*system*]

ARGUMENTS

destination

Specifies a path of system names over which the files are to be transmitted and the login name of a user on the last system who is to receive the files. The *destination* argument has the form

system1 ! [*system2* !]...*login-name*

The value of *system1* must be the name of a remote system in your /usr/lib/uucp/Systems file. Use the uuname command to see the names of systems in that file. For the transfer to complete successfully, the value of *system2* must be configured in the /usr/lib/uucp/Systems file, or its equivalent, on *system1* and so on through the chain of systems.

The value of *login-name* must be the login name of a user on the remote system specified by the last value of *system* in the chain.

file Specifies the file or files that are to be transferred to the remote system. The value of the *file* argument can be the name of a single file; the name of a directory, in which case all the files in the directory hierarchy are transferred; or a string containing shell metacharacters that expand to one or more file or directory names.

-m Sends mail to the user of the uuto command when the transfer to the *system1* is complete.

-p Runs uucp so that it copies files into the spool directory and uses the copy to make the transfer. If you do not use this option, the actual file is read to make the file transfer. In this case, you must not move, delete, or rename the file until the transfer is complete, or the transfer will fail.

-s*system*

Specifies the name of a system when you are using the uupick command and causes uupick to display information about files from the system specified by *system* only. If you do not use this option, uupick displays information about files you have received from all systems.

DESCRIPTION

uuto provides an easy-to-use interface to the uucp command. The files (or directories and files if the *file* argument is a directory) are transferred to a directory named for your system in /usr/spool/uucppublic/receive/*login-name*/*system-name*, where *login-name* is the value of the *login-name* component of the *destination* argument and the value of *system-name* is the name of the system that delivered the file. The user specified by *login-name* is notified by mail when the files arrive.

uupick allows you to accept or dispose of files that have been transmitted by uuto. For each file or directory that uupick finds for you, it displays the following message on the standard output:

from system *system*: [file *file*] [dir *dir*] ?

Then uupick waits for you to enter one of the following disposition commands:

!*command*
: Returns to the shell, which runs the command specified by *command*, and returns to uupick.

* Displays a summary of uupick commands.

a[*dir*]
: Acts the same way as the -m option except that uupick moves all of the files sent from *system.*

CONTROL-D
: Acts the same way as the q command.

d Deletes the file or directory.

m[*dir*]
: Moves the file or directory to the directory specified by *dir*. The value of *dir* can be a full pathname (in which $HOME is allowed) or a relative pathname. If the value of *dir* is a relative pathname, uupick uses your current directory and the value of *dir* as the directory into which to move the file. If you do provide a *dir* argument, uupick uses the current directory.

p Displays the contents of the file.

q Exits uupick.

RETURN
: Causes uuto to go on to next the file or directory.

WARNINGS

To send a file whose name begins with a period, such as `.profile`, you must specify the period rather than relying on the shell to expand a metacharacter for you. For example, the following file specifications are correct:

```
.profile
.prof*
.profil?
```

The following file specifications will not work:

```
*profile
?profile
```

FILES

`/usr/bin/uupick`
Executable file

`/usr/bin/uuto`
Executable file

`/usr/lib/uucp`
Directory of `uucp` commands and configuration files

`/usr/spool/uucppublic`
Directory into which `uuto` transfers files

SEE ALSO

`mail`(1), `uucp`(1C), `uuname`(1C), `uustat`(1C), `uux`(1C)

`uucleanup`(1M) in *A/UX System Administrator's Reference*

NAME

uux — runs a command on a remote system

SYNOPSIS

`uux` [-] [-a*name*] [-b] [-C] [-c] [-g*grade*] [-j] [-n] [-p] [-r] [-s*file*] [-x*debug-level*] [-z] *command-string*

ARGUMENTS

- Causes the standard input to `uux` to be used as the standard input of the command specified by *command-string*. This argument has the same effect as the `-p` option.

-a*login-name*
: Uses the value of *login-name*, rather than using your login name, as the initiator of the `uux` request. If you use this option, notification messages are sent to the user whose login name is *login-name*.

-b Returns in a notification message, the standard input that was provided to the `uux` command if the exit status is nonzero.

-c Copies from the actual file rather than making a copy in the directory `/usr/spool/uucp` and uses that copy to make the transfer. This option is on by default.

-C Makes a copy of a file to be transferred in the directory `/usr/spool/uucp` and uses the copy to make the transfer.

command-string
: Specifies one or more arguments that look like a shell command line, except that the command and filenames may be prepended by *system*`!`, where *system* is the name of a system configured in the `/usr/lib/uucp/Systems` file. If you do not specify a system, `uux` runs the command on your local system.

A filename consists of one of the following:

- A full pathname.
- A pathname preceded by ~*login-name*, where *login-name* is a login name of a user on the specified system and is replaced by the full pathname of that user's login directory.
- A relative pathname, which `uux` converts to a full pathname by prepending the full pathname of the current directory.

-g*grade*
: Specifies a grade. The value of *grade* is a string of one or more alphanumeric characters that specifies a service grade, or priority. To see the available grades, use the `uuglist` command, which displays a list of grades or a message that tells you to use a single letter or number.

`-j` Writes the job identification string on the standard output. You can use the job identification with the `uustat` command to cancel a job or get its status.

`-n` Prevents `uux` from notifying you if the command fails. If you do not use this option, `uux` sends you a mail message describing the failure.

`-p` Causes the standard input to `uux` to be used as the standard input of the command specified by *command-string*. This argument has the same effect as a hyphen (`-`).

`-r` Queues the job but does not start the file transfer.

`-s`*file*
Reports the status of the transfer in the file specified by *file*.

`-x`*debug-level*
Writes debugging information on the standard output. The value of *debug-level* is a number from 0 to 9. Higher numbers produce more detailed debugging information.

`-z` Sends mail when the file transfer completes successfully.

DESCRIPTION

`uux` runs a command on a remote system. You can use `uux` to gather files from specified systems, run a command on the files on a specified system, and then send the standard output of the command to a file on a specified system. For security reasons, some system administrators limit to `mail` the commands that remote `uux` commands can run.

The `uux` command sends mail to notify you if the remote system disallows the requested command. You can disable the notification by using the `-n` option.

EXAMPLES

Here is a command that gets the files `fileA` and `fileB` from `system2` and `system3`, and uses them as arguments to `diff`, which is run on `system1`. The command puts the result of `diff` in the file `file.diff` in the directory `/usr/spool/uucppublic`, which is expanded from the tilde (˜), on `system4`:

```
uux "system1!diff system2!/home/dan/fileA \
             system3!/a4/dan/fileB > \
               !system4~/dan/file.diff"
```

You should quote any special shell characters, such as comma (`,`), right angle bracket (>), semicolon (`;`), and pipe (`|`), either by enclosing the entire command string in quotation marks or by quoting the special characters as individual arguments.

uux attempts to get all appropriate files to the specified system where they are processed. The filenames of output files must be escaped with parentheses. For example, if you are logged in on system1, and want to run uucp on system2 to copy a file from system2 to /tmp on system3, the following uux command would be appropriate:

```
uux "system2!uucp system2!/usr/file (system3!/tmp)"
```

The same command, but without parentheses, will fail because uux will attempt to put system3!/tmp on system2 prior to running uucp on system2.

The following example gets the file /usr/file from system2 and sends it to system1, performs cut on that file, and sends the result to the file /usr/file on system3. Note that parentheses are not required in this example because the only ouput is the result of redirection.

```
uux "system1!cut -f1 system2!/usr/file > system3!/usr/file"
```

LIMITATIONS

Only the first command of a shell pipeline can be preceded by a *system* value. All other commands are executed on the system specified by the first command.

The asterisk shell metacharacter (*) does not expand as you probably want.

The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in a directory known to uucp and its related commands. All files required to run the command are put in this directory unless they already reside on the remote system. You must, therefore, take care to avoid name collisions that result in the overwriting of one file by another. For example, this command fails because the file xyz is first transferred from system2 and then overwritten by the transfer from system3, which results in only one argument for the diff command:

```
uux "system1!diff system2!/home/dan/xyz \
                  system3!/home/dan/xyz > !xyz.diff"
```

If diff is permitted on the remote system, this command does work because the file xyz is transferred to the execution directory from system3, then transferred to system1, and then compared with /home/dan/xyz on system1:

```
uux "system1!diff system1!/home/dan/xyz \
                  system3!/home/dan/xyz > !xyz.diff"
```

You can use uux to transfer protected files that you own and files that are in protected directories that you own. However, if you are logged in as root and your directories cannot be searched by other users, the request will fail.

FILES

`/usr/bin/uux`
: Executable file

`/usr/lib/uucp`
: Directory of `uucp` commands and configuration files

`/usr/lib/uucp/Permissions`
: File that limits the commands that `uux` can run

`/usr/spool/uucp`
: Directory of queued `uucp` jobs

SEE ALSO

`uucp`(1C), `uuglist`(1C), `uustat`(1C)

`uuxqt`(1M) in *A/UX System Administrator's Reference*

Chapter 8, ''Setting Up the UUCP System,'' in *A/UX Network System Administration*

NAME

`val` — validate SCCS file

SYNOPSIS

`val-`

`val` [`-m`*name*] [`-r`*SID*] [`-s`] [`-y`*type*] *file*...

ARGUMENTS

`-` Causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

file Specifies the name of the SCCS file to be validated.

`-m`*name*
Compares the *name* argument value with the SCCS `%M%` keyword in *file*.

`-r`*SID*
Checks to determine if the SCCS delta number, *SID* (*S*CCS *ID*entification String), is is ambiguous (e.g., `-r1` is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (e.g., `-r1.0` or `-r1.1.0` are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.

`-s` Silences the message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

`-y`*type*
Causes `val` to compare the argument value *type* with the SCCS `%Y%` keyword in *file*.

DESCRIPTION

`val` determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to `val` may appear in any order.

`val` generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The 8-bit code returned by `val` is a disjunction of the possible errors, i.e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument
bit 1 = unknown or duplicate keyletter argument
bit 2 = corrupted SCCS file

bit 3 = cannot open file or file not SCCS
bit 4 = *SID* is invalid or ambiguous
bit 5 = *SID* does not exist
bit 6 = `%Y%`, `-y` mismatch
bit 7 = `%M%`, `-m` mismatch

Note that `val` can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned - a logical `OR` of the codes generated for each command line and file processed.

EXAMPLES

```
val  -
-yc -mabc s.abc
-mxyz -ypll s.xyz
```

first checks if file `s.abc` has a value *c* for its `type` flag and value `abc` for the module name flag. Once processing of the first file is completed, `val` then processes the remaining files (in this case `s.xyz`) to determine if they meet the characteristics specified by the keyletter arguments associated with them.

STATUS MESSAGES AND VALUES

Use `help` for explanations.

LIMITATIONS

`val` can process up to 50 files on a single command line. Any number above 50 will produce a `core` dump.

FILES

`/usr/bin/val`
Executable file

SEE ALSO

`admin`(1), `cdc`(1), `comb`(1), `delta`(1), `get`(1), `help`(1), `prs`(1), `rmdel`(1), `sact`(1), `sccs`(1), `sccsdiff`(1), `unget`(1), `what`(1)

`sccsfile`(4) in *A/UX Programmer's Reference*

''SCCS Reference'' in *A/UX Programming Languages and Tools, Volume 2*

See machid(1)

NAME

`vc` — manipulates version control information inside a data stream

SYNOPSIS

`vc` [`-a`] [`-c`*char*] [`-s`] [`-t`] [*keyword=value*]...

ARGUMENTS

`-a` Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in `vc` statements. An uninterpreted control character may be included in a value by preceding it with `\`. If a literal `\` is desired, then it too must be preceded by `\`.

`-c`*char*

Specifies a control character to be used in place of `:`.

keyword=value

Specifies the user declared keyword that was set by `.dcl` and the value it was assisgned. A *keyword* is composed of 9 or less alphanumerics; the first must be alphabetic. A *value* is any ASCII string that can be created with `ed`; a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs. See ''Version Control Statements'' later in this manual page for details.

`-s` Silences warning messages (not error) that are normally printed on the output.

`-t` Ignores all characters from the beginning of a line up to and including the first tab character for the purpose of detecting a control statement. If one is found, all characters up to and including the tabs are discarded.

DESCRIPTION

The `vc` command copies lines from the standard input to the standard output under control of its arguments and control statements encountered in the standard input. In the process of performing the copy operation, user declared *keyword*s may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as `vc` command arguments.

A control statement is a single line beginning with a control character, (unless the `-t` option is used, in which case, all characters up to and including the first tab are ignored, and what follows begins the control statement). The default control character is colon `(:)`. This can be changed by the `-c` option (see below). Input lines beginning with a backslash (`\`) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a noncontrol character are copied in their entirety.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement.

Version Control Statements

`:dcl` *keyword*[, ..., *keyword*]
: Declares keywords. All keywords must be declared.

`:asg` *keyword=value*
: Assigns values to keywords. An `asg` statement overrides the assignment for the corresponding keyword on the `vc` command line and all previous `asg`'s for that keyword. Keywords declared, but not assigned values have null values.

`:if` *condition*

.

.

.

`:end`
: Skips lines of the standard input. If the condition is true all lines between the `if` statement and the matching `end` statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening `if` statements and matching `end` statements are recognized solely for the purpose of maintaining the proper `if-end` matching.

The syntax of a condition is:

```
<cond>   ::= [ "not" ] <or>
<or>     ::= <and> | <and> "|" <or>
<and>    ::= <exp> | <exp> "&" <and>
<exp>    ::= "(" <or> ")" | <value> <op> <value>
<op>     ::= "=" | "!=" | "<" | ">"
<value>  ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than

`( )` used for logical groupings

`not` may only occur immediately after the `if`, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., `: 012 > 12` is false). All other operators take strings as arguments (e.g., `: 012 != 12` is true). The precedence of the operators (from highest to lowest) is:

```
= != > <
        all of equal precedence
&
|
```

Parentheses may be used to alter the order of precedence.

Values must be separated from operators or parentheses by at least one blank or tab.

: : *text*
Replaces keywords on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the `-a` option.

`:on`
`:off`
Turns on or off keyword replacement on all lines.

`:ctl` *char*
Changes the control character to char.

`:msg` ***message***
Prints the given message on the output.

`:err` ***message***
Prints the given message followed by:

```
ERROR: err statement on line ... (915)
```

on the output. `vc` halts execution, and returns an exit code of 1.

EXAMPLES

If you have a file named `note` containing:

```
:dcl NAME,PLACE
        :NAME:,
        Just a note to remind you that we have a meeting
        scheduled Monday morning at :PLACE:.
```

the command

```
vc -a NAME=Joe PLACE=University < note
```

will produce

```
Joe,
Just a note to remind you that we have a meeting
scheduled Monday morning at the University.
```

STATUS MESSAGES AND VALUES

0	normal
1	any error

Use `help` for explanations.

FILES

`/usr/bin/vc`
Executable file

SEE ALSO

`admin`(1), `cdc`(1), `comb`(1), `delta`(1), `ed`(1), `get`(1), `help`(1), `rmdel`(1), `prs`(1), `sact`(1), `sccs`(1), `sccsdiff`(1), `unget`(1), `val`(1), `what`(1)

`sccsfile`(4) in *A/UX Programmer's Reference*

''SCCS Reference'' in *A/UX Programming Languages and Tools, Volume 2*

See vi(1)

NAME

`version` — reports version number of files

SYNOPSIS

`version` *file...*

ARGUMENTS

file Specifies the file for which you want the version number.

DESCRIPTION

`version` takes a list of files and reports the version number. If the file is not a binary, it reports:

```
not a binary
```

If there is not a version number associated with the file, it reports:

```
No version header
```

The `version` command also reports the object file format of the file, such as either

```
Coff object file format
```

or

```
Old a.out object file format.
```

The `version` command is useful for determining which version of the current program you are running.

EXAMPLES

The command

```
version /bin/version
```

prints the version number of the `version` program.

FILES

`/bin/version`
Executable file

SEE ALSO

`strings`(1), `what`(1)

NAME

`vi`, `view`, `vedit` — invokes the screen-oriented (visual) display editor

SYNOPSIS

`vi` [+*command*] [`-l`] [`-r` [*file*]] [`-R`] [`-t` *tag*] [`-w`*n*] [`-x`] *name...*

`view` [+*command*] [`-l`] [`-r` [*file*]] [`-R`] [`-t` *tag*] [`-w`*n*] [`-x`] *name...*

`vedit` [+*command*] [`-l`] [`-r` [*file*]] [`-R`] [`-t` *tag*] [`-w`*n*] [`-x`] *name...*

ARGUMENTS

+command
Interprets the specified `ex` command before editing begins.

`-l` Starts the `LISP` mode; indents appropriately for `lisp` code, the `()`, `{}`, `[[`, and `]]` commands in `vi` and `open` are modified to have meaning for `lisp`.

name
Specifies the name of the file to be edited.

`-r` [*file*]
Recovers *file* after an editor or system crash. If *file* is not specified, a list of all saved files will be printed.

`-R` Starts read only mode; the `readonly` flag is set, preventing accidental overwriting of the file.

`-t` *tag*
Edits the file containing the *tag* and positions the editor at its definition.

`-w`*n*
Sets the default window size to *n*. This is useful when using the editor over a slow speed line.

`-x` Starts encryption mode; a key is prompted for allowing creation or editing of an encrypted file. This encryption scheme is not secure.

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 file `$HOME/.exrc` and the variable `EXINIT` can be used to establish preferences (initializations) that take effect whenever you run `vi` or `ex`. For example, to invoke line numbering mode automatically you could place the following string in the `exrc` file or assign it as a quoted argument to `EXINIT`, which you would then export:

```
set number
```

When using `vi`, changes you make to the file are reflected by what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

The *name* argument indicates files to be edited.

The `view` invocation is the same as `vi` except that the `readonly` flag is set.

The `vedit` invocation is intended for beginners. The `report` flag is set to 1, and the `showmode` and `novice` flags are set. These defaults make it easier to initially learn the editor.

Modes

Command
: Normal and initial mode. Other modes return to command mode upon completion. ESCAPE is used to cancel a partial command.

Input
: Entered by an `a`, `i`, `A`, `I`, `o`, `O`, `c`, `C`, `s`, `S`, or `R`. Text may then be entered. Input mode is normally terminated with ESC character, or abnormally with interrupt.

Last line
: Reading input for `:`, `/`, `?` or `!`; terminate with RETURN to execute, interrupt to cancel.

Sample Commands

$\leftarrow \downarrow \uparrow \rightarrow$
: arrow keys move the cursor

`h j k l`
: same as arrow keys

`i`*text* ESCAPE
: insert *text*

`cw`*new* ESCAPE
: change word to *new*

`eas`ESCAPE
: pluralize word

`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!`RETURN
: quit, discarding changes

/*text* RETURN
: search for *text*

CONTROL-U, CONTROL-D
: scroll up or down

:*ex-cmd*RETURN
: 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
: CONTROL-D CONTROL-U

repeat effect
: most of the rest

Interrupting and Canceling

ESCAPE
: ends insert mode or abandons a partially-entered command

interrupt
: interrupts any command that is still underway

CONTROL-L
: reprints screen

CONTROL-R
: reprints screen if CONTROL-L is → key

File Manipulation

:wRETURN
: write back changes

:qRETURN
: quit

`:q!`RETURN
: quit, discard changes

:e*name*RETURN
: edit file *name*

`:e!`RETURN
: reedit, discard changes

`:e+`*name*RETURN
: edit, starting at end

`:e+`*n*RETURN
: edit starting at line *n*

`:e#`RETURN
: edit alternate file

`:w`*name*RETURN
: write file *name*

BI :w! name RETURN
: overwrite file *name*

`:sh`RETURN
: run shell, then return

`:!`*cmd*RETURN
: run *cmd*, then return

`:n`RETURN
: edit next file in arglist

`:n`*args*RETURN
: specify new arglist

CONTROL-G
: show current file and line

`:ta`*tag*RETURN
: tag file entry *tag*

In general, any `ex` or `ed` command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a RETURN.

Positioning Within a File

CONTROL-F
: forward screen

CONTROL-B
: backward screen

CONTROL-D
: scroll down half screen

CONTROL-U
: scroll up half screen

`G` go to specified line (end default)

/pat
: next line matching *pat*

?*pat*
: prev line matching *pat*

n
: repeat last / or ?

N
: reverse last / or ?

/*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

CONTROL-L
: clear and redraw

CONTROL-R
: retype, eliminate @ lines

zRETURN
: redraw, current at window top

z-RETURN
: ... at bottom

z.RETURN

/*pat*/z-RETURN
: *pat* line at bottom

z*n*.RETURN
: use *n* line window

CONTROL-E
: scroll window down 1 line

CONTROL-Y
: scroll window up 1 line

Marking and Returning

‘ ‘	move cursor to previous context
’ ’	... at first nonwhite in line
m*x*	mark current position with letter *x*
'*x*	move cursor to mark *x*
'*x*	... at first nonwhite in line

Line Positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first nonwhite
-	previous line, at first nonwhite
RETURN	return, same as +
↓ j	next line, same column
↑ k	previous line, same column

Character Positioning

^	first nonwhite
0	beginning of line
$	end of line
h →	forward
l ←	backward
CONTROL-H	same as ←
space	same as →
f*x*	find *x* forward
F*x*	same as f, but in the backward direction
t*x*	moves cursor forward to x
T*x*	back up to x

`;`	repeat last `f` `F` `t` or `T`
`,`	inverse of `;`
`|`	to specified column
`%`	find matching `(` `{` `)` or `}`

Words, Sentences, and Paragraphs

`w`	word forward
`b`	back word
`e`	end of word
`)`	to next sentence
`}`	to next paragraph
`(`	back sentence
`{`	back paragraph
`W`	blank delimited word
`B`	back `W`
`E`	to end of `W`

Commands for LISP Mode

`)`	Forward *s-expression*
`}`	... but do not stop at atoms
`(`	Back *s-expression*
`{`	... but do not stop at atoms

Corrections During Insert

CONTROL-H	erase last character
CONTROL-W	erase last word
erase	your erase, same as CONTROL-H
kill	your kill, erase input this line
\	quotes CONTROL-H, your erase and kill
ESCAPE	ends insertion, back to command
interrupt	your interrupt, terminates insert

CONTROL-D
: backtab over *autoindent*

↑CONTROL-D
: kill *autoindent*, save for next

0CONTROL-D
: ... but at margin next also

CONTROL-V
: quote nonprinting character

Insert and Replace

`a`	append after cursor
`i`	insert before cursor
`A`	append at end of line
`I`	insert before first nonblank
`o`	open line below
`O`	open line above
`r`*x*	replace single character with *x*
`R`*text*ESCAPE	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, for example, `dd` to affect whole lines.

`d`	delete
`c`	change
`y`	yank lines to buffer
`<`	left shift
`>`	right shift
`!`	filter through command
`=`	indent for LISP

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`	... before cursor (`dh`)
`Y`	yank lines (`yy`)

Yank and Put

put inserts the text most recently *deleted* or *yanked*. If a buffer is named, however, the text in that buffer is *put* instead.

`p`	put back text after cursor
`P`	put text before cursor
`"`*x*`p`	put text from buffer *x*
`"`*x*`y`	yank text to buffer *x*
`"`*x*`d`	delete text into buffer *x*

Undo, Redo, and Retrieve

`u`	undo last change
`U`	restore current line
`.`	repeat last change
`"`*d*`p`	retrieve *d*th last delete

LIMITATIONS

Software tabs using CONTROL-T work immediately after the `autoindent` only.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

FILES

`$HOME/.exrc`
: File containing `ex` initialization commands

`/usr/bin/vi`
: Executable file

`/usr/bin/view`
: Executable file

`/usr/bin/vedit`
: Executable file

SEE ALSO

ex(1)

“Using vi,” in *A/UX Text Editing Tools*

See vi(1)

NAME

`w` — displays a summary of the current system activity

SYNOPSIS

`w` [`-h`] [`-l`] [`-s`] [`-u`] [*user*]

ARGUMENTS

`-h` Suppresses the heading.

`-l` Gives the long output, which is the default.

`-s` Asks for a short form of output. In the short form, the tty is abbreviated, the login time and CPU times are left off, as are the arguments to commands.

`-u` Suppresses everything but the heading, as in `uptime`(1).

user
Causes the output to be restricted to the specified user.

DESCRIPTION

`w` prints a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields output are: the users login name, the name of the tty the user is on, the time of day the user logged on, the number of minutes since the user last typed anything, the CPU time used by all processes and their children on that terminal, the CPU time used by the currently active processes, the name and arguments of the current process.

LIMITATIONS

The notion of the current process is muddy. The current algorithm selects the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal. This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. (In cases where no process can be found, `w` prints `-`.)

The CPU time is only an estimate; in particular, if someone leaves a background process running after logging out, the person currently on that terminal is `charged` with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

The `w` command does not know about the new conventions for detection of background jobs. It will sometimes find a background job instead of the right one.

FILES

`/usr/ucb/w`
Executable file

`/etc/utmp`
File containing user status information

`/dev/kmem`
Device file

SEE ALSO

`who`(1), `finger`(1), `ps`(1), `uptime`(1), `users`(1)

`utmp`(4) in *A/UX Programmer's Reference*

NAME

`wc` — counts characters, words, and lines in a file

SYNOPSIS

`wc` [-[*chunk-size*]] [*file*]...

ARGUMENTS

-[*chunk-size*]

Specifies one or more things to count. Replace *chunk-size* with one or a combination of the following options:

`c`	Requests a report of the number of characters in a file.
`l`	Requests a report of the number of lines in a file.
`w`	Requests a report of the number of words in a file.

file Specifies the *file* to be counted.

DESCRIPTION

`wc` counts lines, words, and characters in the specified *files* or in the standard input (if no *files* appear). It also keeps a total count for all *files.* A word is a maximal string of characters delimited by spaces, tabs, or newlines.

The options may be used in any combination; the default is `-lwc`.

When *file*s are specified on the command line, they will be printed along with the counts.

EXAMPLES

Enter this command:

```
wc filea fileb filec
```

to report the number of lines, words, and characters in each of the files, as well as the totals.

FILES

`/bin/wc`

Executable file

SEE ALSO

`sum`(1), `sumdir`(1)

NAME

`what` — reports identification information for a file

SYNOPSIS

`what [-s]` *file...*

ARGUMENTS

file	Specifies the file to be searched.
`-s`	Quits after finding the first occurrence of pattern in each file.

DESCRIPTION

`what` searches the given files for all occurrences of the pattern that `get` substitutes for `%Z%` (this is `@(#)` at this printing) and prints out what follows until the first `' '`, >, newline, \, or null character. For example, if the C program in file `f.c` contains

char ident[] = `''@ (#)`
identification information'';

and `f.c` is compiled to yield `f.o` and `a.out`, then the command

```
what f.c f.o a.out
```

will print

```
f.c:    identification information
f.o:    identification information
a.out:
        identification information
```

The `what` program is intended to be used in conjunction with the SCCS command `get`, which inserts identifying information automatically, but it can also be used where the information is inserted manually.

EXAMPLES

If `test1.c` has the following string:

```
char v[] = "@(#)1 test1.c 2";
```

typing

```
what test1.c
```

would print the following:

```
test1.c:
        1 test1.c 2
```

STATUS MESSAGES AND VALUES

The exit status is 0 if any matches are found, otherwise it is 1. Use the `help` command for explanations.

LIMITATIONS

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

FILES

`/usr/bin/what`
Executable file

SEE ALSO

admin(1), cdc(1), comb(1), delta(1), get(1), help(1), prs(1), rmdel(1), sact(1), sccs(1), sccsdiff(1), unget(1), val(1)

sccsfile(4) in *A/UX Programmer's Reference*

''SCCS Reference'' in *A/UX Programming Languages and Tools, Volume 2*

NAME

`whatis` — reports a brief description for the manual page entry specified

SYNOPSIS

`whatis` *command...*

ARGUMENTS

command

Specifies the command you wish to display information about.

DESCRIPTION

`whatis` looks up a given command and gives the header line from the manual section. You may then run the `man`(1) command to get more information.

EXAMPLES

If you type:

```
whatis ed
```

you will see:

```
red ed(1)          - text editor
```

This tells you that the *section* for `ed` is 1. To see the entire manual entry for `ed`, on the terminal, enter:

```
man 1 ed
```

FILES

`/usr/ucb/whatis`

Executable file

`/usr/lib/whatis`

Executable file

SEE ALSO

`apropos`(1), `man`(1), `whereis`(1), `which`(1)

NAME

whereis — reports the locations of the source, binary, and online help files for a specified command

SYNOPSIS

whereis [-b] [-B *dir* [-f]] [-m] [-M *dir* [-f]] [-s] [-S *dir* [-f]] [-u] *file...*

ARGUMENTS

-b Searches only for binaries.

-B *dir*
Searches in the specified directory, *dir*, for binaries.

-f Terminates the last specified directory list and signals the start of the filenames.

-m Searches only for manual sections.

-M *dir*
Searches in the specified directory, *dir*, for manual sections.

file Specifies the *file* to be located.

-s Searches only for sources.

-S *dir* [-f]
Searches in the specified directory, *dir*, for sources.

-u Searches for unusual entries. A file is considered unusual if it does not have one entry of each requested type.

DESCRIPTION

whereis locates source, binary, and manual sections for specified files. The supplied *file*(*s*) are first stripped of leading pathname components and standard extensions for source files and manual entries (such as .c and .1m). Prefixes of s. resulting from use of source code control are also dealt with. whereis then attempts to locate the desired program in a list of standard places.

EXAMPLES

To ask for those files in the current directory which have no documentation, enter:

```
whereis -m -u *
```

The following finds all the files in /usr/bin which are not documented in /usr/man/man1 with source in /usr/src/cmd:

```
cd /usr/ucb
whereis -u -M /usr/man/man1 -S /usr/src/cmd -f *
```

LIMITATIONS

Since the program uses `chdir` to run faster, pathnames given with the `-M`, `-S`, and `-B` must be full; that is, they must begin with a `/`.

FILES

`/usr/bin/whereis`
  Executable file

`/usr/src/*`
  Source files

`/usr/catman/man/*`
  Files containing on-line manual pages

`/bin`
  Directory containing administrative binary files

`/lib`
  Directory containing administrative library files

`/etc`
  Directory containing administrative executable command files

`/usr/bin`
  Directory containing user binary files

`/usr/lib`
  Directory containing user library files

`/usr/etc`
  Directory containing user executable command files

`/usr/ucb`
  Directory containing user executable files

SEE ALSO

`whatis`(1), `which`(1)

NAME

which — reports the directory path to a file by interpreting PATH and alias settings

SYNOPSIS

which [*name*]...

ARGUMENTS

name

Specifies the name that which searches for the executable file(s).

DESCRIPTION

which takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and is searched for along the user's path. Both aliases and paths are taken from the user's .cshrc file.

STATUS MESSAGES AND VALUES

A value is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

NOTES

which Operates only with csh.

FILES

/usr/ucb/which

Executable file

~/.cshrc

Source file containing aliases and path values

SEE ALSO

whereis(1), which(1)

NAME

`who` — reports users who are currently logged in to the system

SYNOPSIS

`who` [`-a`] [`-b`] [`-d`] [`-H`] [`-l`] [`-p`] [`-s`] [`-t`] [`-T`] [`-u`] [*file*]

`who -r` [`-d`] [`-l`] [`-p`] [`-u`] [*file*]

`who -q` [*file*]

`who am i`

`who am I`

ARGUMENTS

`-a` Processes `/etc/utmp` or the file specified by the *file* argument, with all options turned on.

`-b` Indicates the time and date of the last reboot.

`-d` Displays all processes that have expired and have not been respawned by `init`.

file Specifies the file to be examined. Usually, *file* is `/etc/wtmp` which contains a history of all the logins since the file was last created.

`-H` Prints column headings above the regular output.

`-l` Lists only those lines on which the system is waiting for someone to log in. The *name* field (described in the ''Description'' later in this manual page) contains `LOGIN` in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

`-p` Lists any non-`getty` process that is currently active and has been previously spawned by `init`. The *name* field contains the name of the program executed by `init` as found in `/etc/inittab`. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from `/etc/inittab` that spawned this process. See `inittab`(4). The *exit* field appears for dead processes and contains the termination and exit values (as returned by `wait`(2)) of the dead process. This option can be useful in determining why a process terminated.

`-q` Displays only the names and the number of users currently logged in. When this option is used, all other options are ignored.

`-r` Indicates the current run-level of the `init` process as part of an expanded listing of the system status. If the run-level shown is 2, then the system is currently running in multi-user mode.

`-s` Lists only the *name*, *line*, *time,* and *remote host* (if any) fields. This option is the default.

`-t` Indicates the last change to the system clock (by means of the `date`(1) command) by a user logged in as `root`. See `su`(1).

`-T` Does the same thing as the `-u` option, and also prints the *state* field of the terminal line. The *state* field describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a - appears if it is not. A user logged in as `root` can write to all lines having a + or a - in the `state` field. If a bad line is encountered, a `?` is printed.

`-u` Lists only those users who are currently logged in. The *name* field contains the user's login name. The *line* field contains the name of the line as found in the directory `/dev`. The *time* field contains the time that the user logged in. The *activity* field 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 24 hours have elapsed or the line has not been used since boot time, the entry is marked `old`. This field is useful when you are trying to determine whether a person is working at the terminal or not. The *pid* field contains the process ID of the user's shell. The *comment* field contains the comment field associated with this line as found in `/etc/inittab` (see `inittab`(4)). This field can contain information about where the terminal is located, the telephone number of the dataset, the type of terminal if hard-wired, and so on.

DESCRIPTION

`who` can list the user's login name, the terminal line, the login time, the elapsed time since activity occurred on the line, and the process ID of the command interpreter (shell) for each current A/UX system user. It examines the `/etc/utmp` file to obtain its information. If *file* is given, that file is examined. Usually, *file* will be `/etc/wtmp`, which contains a history of all the logins since the file was last created.

If the `who` command is issued with `am i` or `am I` at the end, the command identifies the invoking user.

Except for the default `-s` option, the general format for output entries is as follows:

name [*state*] *line time activity pid* [*comment*] [*exit*]

With options, `who` can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the `init` process.

EXAMPLES

The following command reports the name under which you are currently logged in.

```
who am i
```

FILES

`/bin/who`
: Executable file

`/etc/inittab`
: Initialization table file

`/etc/utmp`
: Temporary file

`/etc/wtmp`
: Temporary file

SEE ALSO

`date`(1), `login`(1), `mesg`(1), `ps`(1), `su`(1), `users`(1), `w`(1), `whoami`(1)

`wait`(2), `inittab`(4), `utmp`(4) in *A/UX Programmer's Reference*

`init`(1M) in *A/UX System Administrator's Reference*

NAME

`whoami` — prints effective current user ID

SYNOPSIS

`whoami`

DESCRIPTION

`whoami` prints who you are. It works even if you are logged in as superuser, while `who am i` does not since it uses `/etc/utmp`.

FILES

`/usr/bin/whoami`
  Executable file
`/etc/passwd`
  Name data base file

SEE ALSO

`id`(1), `who`(1)

NAME

`write` — writes to another user

SYNOPSIS

`write` *user* [*line*]

ARGUMENTS

line	Indicates which line or terminal (such as (`tty00`)) to send the message to. If this argument is not specified, 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*
user	Specifies the name of the person you wish to send a message to.

DESCRIPTION

`write` copies lines from your terminal to that of another user. Writing to others is normally allowed by default. Certain commands, in particular `nroff`(1) and `pr`(1) prevent messages from being sent to avoid interference with their output. However, if the user has superuser 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.

Permission to write may be denied or granted by use of the `mesg` command.

The following protocol is suggested for using the `write` command:

- 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.

EXAMPLES

If you want to write to a user account named `cheryl` at terminal (tty) number 2, enter:

```
write cheryl tty2
```

If Cheryl has her terminal set to receive messages with `mesg y` (refer to `mesg`(1)), she will receive the message:

`Message from` *yourname* `(tty??)` [*date*]...

where `tty??` is your terminal number and *date* is the time and date the message was sent. Once the connection is successful, two bells are sent to your terminal which indicates that the data you are typing is being sent.

The user, Cheryl, should write back at this point. Communication can continue until an end of file is read from the terminal, an interrupt is sent from either user, or the recipient (Cheryl, in this example) has executed the `mesg n` command. When this occurs, `write` writes `EOT` on the other terminal and exits.

STATUS MESSAGES AND VALUES

`user is not logged in`
: The person you are trying to `write` to is not logged in.

`Permission denied`
: The person you are trying to `write` to denies that permission with the `mesg` command.

`Warning: cannot respond, set mesg -y`
: Your terminal is set to `mesg n` and the recipient cannot respond to you.

`Can no longer write to user`
: The recipient has denied permission with the `mesg n` command after you started writing.

FILES

`/bin/write`
: Executable file

`/etc/utmp`
: Temporary file

`/bin/sh`
: Shell file

SEE ALSO

`mail`(1), `mesg`(1), `nroff`(1), `pr`(1), `sh`(1), `talk`(1N), `who`(1)

`wall`(1M) in *A/UX System Administrator's Reference*

NAME

`xargs` — builds arguments based on the standard input, passing them in batches to the specified command which is executed enough times to deplete all the arguments

SYNOPSIS

`xargs` [`-e`*eofstr*] [`-i`*replstr*] [`-l`*number*] [`-n`*number*] [`-p`] [`-s`*size*] [`-t`] [`-x`] [*command* [*cmd-args*]]

ARGUMENTS

command

Specifies the given *command* which `xargs` executes.

cmd-args

Represents the arguments that are passed to the specified *command* when it is executed.

`-e`*eofstr*

Specifies 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.

`-i`*replstr*

Indicates the insert mode. The *command* is executed for each line from standard input, taking the entire line as a single argument, inserting it in *cmd-args* for each occurrence of *replstr*. A maximum of 5 arguments in *cmd-args* 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 the `-x` option is forced. `{}` is assumed for *replstr* if not specified.

`-l`*number*

Executes *command* for each nonempty *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 newline *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next nonempty line. If *number* is omitted, 1 is assumed. The `-x` option is forced.

-n*number*
: Executes *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 the -x option is also coded, each *number* arguments must fit in the *size* limitation, else xargs terminates execution.

-p
: Specifies the prompt mode. The user is asked whether to execute *command* at 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*.

-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 1180. If -s is not coded, 1180 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.

-t
: Indicates the trace mode. The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

-x
: Causes xargs to terminate if any argument list would be greater than *size* characters; -x is forced by the -i and -l options. When none of the -i, -l, or -n options are coded, the total length of all arguments must be within the *size* limit.

DESCRIPTION

xargs combines the fixed *cmd-args* 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 options specified.

The replacement for *command*, may be a shell file. In any case, xargs searches for commands as directed by the $PATH variable. 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 newlines; 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 *cmd-args*, followed by some number of arguments read from standard input (Exception: see the `-i` option). The `-i`, `-l`, and `-n` options determine how arguments are selected for each command invocation. When none of these options are coded, *cmd-args* 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 option conflicts (for example, `-l` versus `-n`), the last option has precedence.

The `xargs` program will terminate if: it cannot execute *command*; or if *command* returns a -1 exit status.

When *command* is a shell program, it should explicitly *exit* (see `sh`(1)) with an appropriate value to avoid accidentally returning with -1 .

EXAMPLES

Enter this command:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

to move all files from directory `$1` to directory `$2`, and echo each move command just before doing it.

Entering the command:

```
(logname; date; echo $0 $*) | xargs >>log
```

will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file `log`.

This command:

```
ls | xargs -p -l ar r arch
ls | xargs -p -l | xargs ar r arch
```

causes the user to be asked which files in the current directory are to be archived and archives them into `arch` one at a time in the first instance, or as in the second instance, many at a time.

Enter:

```
echo $* | xargs -n2 diff
```

to execute `diff`(1) with successive pairs of arguments originally typed as shell arguments.

FILES

`/usr/bin/xargs`
Executable file

SEE ALSO

csh(1), ksh(1), sh(1)

NAME

`xstr` — reports strings from C programs to implement shared strings

SYNOPSIS

`xstr [-] [-c]` [*file*]

ARGUMENTS

- `-` Causes `xstr` to read from its standard input.

- `-c` Extracts the strings from the C source program into *file*, replacing string references by expressions of the form (`&xstr` [*number*])for some number. An appropriate declaration of `xstr` is prefixed to *file*. The resulting C text is placed in the file `x.c`, to then be compiled. The strings from *file* are placed in the *strings* data base if they are not already there. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

- *file* Specifies the name of the file that the strings will be extracted from.

DESCRIPTION

`xstr` maintains a file called `strings` into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

After all components of a large program have been compiled, a file `xs.c` declaring the common `xstr` space can be created by a command of the form

```
xstr
```

This `xs.c` file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

It may be useful to run `xstr` after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed.

EXAMPLES

An appropriate command sequence for running `xstr` after the C preprocessor is:

```
cc -E file.c | xstr -c -
cc -c x.c
mv x.o file.o
```

The `xstr` program does not touch the file `strings` unless new items are added, thus `make` can avoid remaking `xs.o` unless truly necessary.

The `xstr` program can also be used on a single file. A command

`xstr` *file*

creates files `x.c` and `xs.c` as before, without using or affecting any `strings` file in the same directory.

STATUS MESSAGES AND VALUES

If a string is a suffix of another string in the data base, but the shorter string is seen first by `xstr` both strings will be placed in the data base, when just placing the longer one there will do.

FILES

`/usr/bin/xstr`	Executable file
`strings`	Data base of strings
`x.c`	Massaged C source file
`xs.c`	C source file for definition of array `xstr`
`/tmp/xs*`	Temporary file when ``xstr` *file*''doesn't touch `strings`

SEE ALSO

`mkstr`(1), `strings`(1)

NAME

yacc — compiles compilers (yet another compiler-compiler)

SYNOPSIS

yacc [-d] [-l] [-t] [-v] *grammar*

ARGUMENTS

-d Generates the y.tab.h file with the #define statements that associate the yacc-assigned ''token codes'' with the user-declared ''token names.'' This allows source files other than y.tab.c to access the token codes.

grammar
Specifies the data that will be converted by yacc.

-l Prevents the code produced in y.tab.c from containing any #line constructs. This option should only be used after the *grammar* and the associated actions are fully debugged.

-t Compiles the runtime debugging code.

-v Prepares the y.output file which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the *grammar*.

DESCRIPTION

yacc converts a context-free *grammar* into a set of tables for a simple automaton which executes an lr parsing algorithm. The *grammar* may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, y.tab.c, must be compiled by the C compiler to produce a program yyparse. This program must be loaded with the lexical analyzer program, yylex, as well as main and yyerror, an error handling routine. These routines must be supplied by the user. The lex routine is useful for creating lexical analyzers usable by yacc.

Runtime debugging code is always generated in y.tab.c under conditional compilation control. By default, this code is not included when y.tab.c is compiled. However, when yacc's -t option is used, this debugging code will be compiled by default. Independent of whether the -t option was used, the runtime debugging code is under the control of YYDEBUG, a pre-processor symbol. If YYDEBUG has a nonzero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

EXAMPLES

To process a file called file1.y in yacc format, enter:

```
yacc file1.y
```

STATUS MESSAGES AND VALUES

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the `y.output` file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

LIMITATIONS

Because filenames are fixed, only one `yacc` process can be active in a given directory at a time.

FILES

`/bin/yacc`
: Executable file

`y.output`
: Output file

`y.tab.c`
: File

`y.tab.h`
: File

`yacc.tmp`
: Temporary file

`yacc.debug`
: Temporary file

`yacc.acts`
: Temporary file

`/usr/lib/yaccpar`
: Temporary file

SEE ALSO

`lex`(1)

`malloc`(3X) in *A/UX Programmer's Reference*

The yacc reference in *A/UX Programming Languages and Tools, Volume 2*

NAME

`yes` — generates `y` entries in response to requests for input

SYNOPSIS

`yes` [*expletive*]

ARGUMENTS

[*expletive*]

Outputs *expletive* repeatedly. If *expletive* is a multiple word, the words must be enclosed in quotes.

DESCRIPTION

`yes` repeatedly outputs `y`. Termination is by the interrupt character (by default, CONTROL-c in the A/UX standard distribution).

FILES

`/usr/ucb/yes`

Executable file

NAME

ypcat — lists the contents of a Network Information Service (NIS) map

SYNOPSIS

ypcat [-d *domain-name*] [-k] [-t] *map-or-nick-name*

ypcat -x

ARGUMENTS

-d *domain-name*
: Specifies a domain other than the local system's domain as returned by the domainname command.

-k Displays each key and its corresponding value.

-t Inhibits the translation of a nickname to a map name. For example, ypcat -t passwd fails because there is no map named passwd, whereas the passwd in ypcat passwd is translated to ypcat passwd.byname.

-x Displays the map nickname table. The table lists the nicknames the command knows of and indicates the long map name associated with each nickname.

DESCRIPTION

ypcat lists the contents of the NIS map specified by *map-or-nick-name*, which may be either the name of a map or a map's ``nickname.'' The ypcat command uses the NIS service to determine the server from which to get the map.

The most common use of NIS is to provide lookup service for administrative files such as the /etc/passwd file. When a system is managed by NIS, each system on the network has a local /etc/passwd file that contains required administrative entries. The master server maintains a global password file containing a login entry for each user of a system on the network. The global password file is used to generate a file that in NIS terminology is called a map. The NIS subsystem uses the map to provide information when commands, running on an NIS client, request NIS service.

The information in maps is sorted in a variety of ways. For example, one of the password maps is passwd.byname in which the entries are sorted by login name.

The map name passwd.byname is an example of a long map name. To accommodate the System V file system, which limits names to 14 characters, the A/UX® implementation of NIS uses short map names and translates any long map names into short map names. In addition to long map names and short map names, the most commonly used maps have nicknames. Here is a list of nicknames, long map names, short map names, and the source files they correspond to:

Nickname	**Long map name**	**Short map name**	**Source file**
`aliases`	`mail.aliases`	`m.a`	`/usr/lib/aliases`
`ethers`	`ethers.byname`	`e.nm`	`/etc/ethers`
`group`	`group.byname`	`grp.nm`	`/etc/group`
`hosts`	`hosts.byaddr`	`hst.ad`	`/etc/hosts`
`passwd`	`passwd.byname`	`pwd.nm`	`/etc/passwd`
`protocols`	`protocols.bynumber`	`ptc.nr`	`/etc/protocols`
	`netgroup`	`netg`	`/etc/netgroup`
`networks`	`networks.byaddr`	`ntw.ad`	`/etc/networks`
`services`	`services.byname`	`svc.nm`	`/etc/services`

Refer to `ypfiles`(4) and `ypserv`(1M) for an overview of NIS.

EXAMPLES

To see the contents of the global password file, enter:

```
ypcat passwd
```

FILES

`/usr/bin/ypcat`
Executable file

SEE ALSO

`domainname`(1), `ypmatch`(1)

`ypserv`(1M) in *A/UX System Administrator's Reference*

`ypfiles`(4) in *A/UX Programmer's Reference*

Chapter 4, ''Setting Up the Network Information Service,'' in *A/UX Network System Administration*

NAME

`ypmatch` — lists the value of a specified key in a Network Information Service (NIS) map

SYNOPSIS

`ypmatch` [`-d` *domain*] [`-k`] [`-t`] *key ... nickname-or-map-name*

`ypmatch -x`

ARGUMENTS

`-d` *domain*
: Specifies a domain other than the local system's domain as returned by the `domainname` command.

`-k` Causes `ypmatch` to display the value of the *key* argument, followed by a colon (:), before displaying the value of the key itself. This option is useful only if the keys are not duplicated in the values, or if you've specified so many keys that the output could be confusing.

`-t` Inhibits the translation of a nickname to a map name. For example, `ypcat -t passwd` fails because there is no map named `passwd`, whereas the `passwd` string in `ypcat passwd` is translated to `ypcat passwd.byname`.

`-x` Displays the map nickname table. The table lists the nicknames the command knows of and indicates the long map name associated with each nickname.

DESCRIPTION

`ypmatch` lists the values associated with one or more keys from the NIS map specified by a *nickname-or-map-name*, which may be the nickname, long name, or short name of a map. See `ypcat(1)` for a discussion of map names and their values.

When you specify multiple keys, the same map is searched for each key. The keys must be exact values insofar as capitalization and length are concerned. Pattern matching is not available.

STATUS MESSAGES AND VALUES

If `ypmatch` cannot match a key, the following message is displayed:

```
Can't match key.  Reason: no such key in map.
```

FILES

`/usr/bin/ypmatch`
: Executable file

SEE ALSO

`ypcat(1)`

ypfiles(4) in *A/UX Programmer's Reference*

Chapter 4, ''Setting Up the Network Information Service,'' in *A/UX Network System Administration*

NAME

`yppasswd` — changes a login password on the Network Information Service (NIS) master server

SYNOPSIS

`yppasswd` [*login-name*]

ARGUMENTS

login-name

Specifies the login name of the user whose password is to be changed. If you do not specify a value for *login-name*, `yppasswd` attempts to change your password.

DESCRIPTION

`yppasswd` adds or changes a password in the `/etc/passwd` file on the NIS master server. The master server must be running the password daemon, `yppasswdd`, for this command to work.

Before changing an existing password, `yppasswd` prompts for the old password, as stored in the `/etc/passwd` file on the NIS master server, and then prompts twice for the new password to verify your entry.

New passwords can have a minimum of four characters if they use a sufficiently rich alphabet and must have at least six characters if all of the characters are in the same case. These rules are discarded if you enter the same password often enough.

A normal user can change his or her password only. A user who is logged in as `root` can change the password of any user. In either case, `yppasswd` prompts for the old password.

LIMITATIONS

The `yppasswd` command uses an update protocol that passes all the information to `yppasswdd` in one remote procedure call, without ever looking at it. Thus, if you type your old password incorrectly, you are not notified until after you have entered and verified your new password.

FILES

`/usr/bin/yppasswd`

Executable file

SEE ALSO

`passwd(1)`

`yppasswdd(1M)` in *A/UX Programmer's Reference*

`ypfiles(4)` in *A/UX System Administrator's Reference*

Chapter 4, ''Setting Up the Network Information Service,'' in *A/UX Network System Administration*

NAME

`ypwhich` — displays the host name of a system's Network Information Service (NIS) server

SYNOPSIS

`ypwhich` [`-d` *domain-name*] [`-V1`] [*host-name*]

`ypwhich` [`-d` *domain-name*] [`-V2`] [*host-name*]

`ypwhich` [`-d` *domain-name*] [[`-t`] `-m` [*map-or-nickname*]]

`ypwhich -x`

ARGUMENTS

`-d` *domain-name*
: Specifies a domain other than the local system's domain as returned by the `domainname` command.

host-name
: Specifies the name of a system to ask about rather than the local system. This argument can be used only with the `-d`, `-V1`, and `-V2` options.

`-m` [*map-or-nickname*]
: Displays the NIS master server for a map. The value of *map-or-nickname* can be a nickname, a long map name, or a short map name. See `ypcat`(1) for a discussion of map names and their values. If you do not specify a *map-or-nickname* argument, `ypwhich` produces a list of available maps and their servers. You cannot specify a *host-name* argument with this option.

`-t` *map-name*
: Inhibits the translation of a nickname to a map name. The value of *map-name* can be a long or short map name. See `ypcat`(1) for a discussion of map names and their values. This option is useful if a map name is identical to a nickname.

`-V1`
: Displays the host name of a server that is serving client processing which uses the version 1 NIS protocol. The version 1 protocol is the old version of the protocol.

`-V2`
: Displays the host name of a server that is serving client processing which uses the version 2 NIS protocol. The version 2 protocol is the current version of the protocol.

`-x` Displays the map nickname table. The table lists the nicknames the command knows of and indicates the long map name associated with each nickname.

DESCRIPTION

`ypwhich` displays the host name of the NIS server that is supplying NIS services to an NIS client or displays the system that is the NIS master server for a map. If run without any argument, `ypwhich` displays the name of the NIS server for the local system. If you specify *host-name*, `ypwhich` displays the name of that system's NIS master server.

If you do not use the `-V1` or `-V2` option, `ypwhich` attempts to locate the server that supplies version 2 services. If no version 2 server is currently found, `ypwhich` attempts to locate the server supplying version 1 services. Because NIS servers and clients are backward-compatible, you seldom need to be concerned about which version is currently in use.

Refer to `ypfiles`(4) and `ypserv`(1M) for an overview of the NIS.

FILES

`/usr/bin/ypwhich`
    Executable file

SEE ALSO

`rpcinfo`(1M), `ypserv`(1M), `ypset`(1M) in *A/UX System Administrator's Reference*

`ypfiles`(4) in *A/UX Programmer's Reference*

Chapter 4, ''Setting Up the Network Information Service,'' in *A/UX Network System Administration*

See compress(1)

See compress(1)

See compress(1)

See compress(1)

Table of Contents

Section 6: Games

NAME

`intro` — introduction to games

DESCRIPTION

This section describes the recreational and educational programs found in the `/usr/games` directory.

NAME

`aliens` — plays the game of Space Invaders (A/UX version)

SYNOPSIS

`aliens`

DESCRIPTION

aliens is the A/UX version of Space Invaders, where alien invaders attack the earth. The program is pretty much self-documenting.

LIMITATIONS

The program is a CPU hog. It needs to be rewritten. It doesn't do well on terminals that run slower than 9600 baud.

FILES

`/usr/games/aliens`
    Executable file

`/usr/games/alienslog`
    Log file

NAME

`arithmetic` — provides arithmetic problems

SYNOPSIS

`arithmetic` [+] [-] [x] [/] [*range*]

ARGUMENTS

+	Causes addition problems to be generated.
-	Causes subtraction problems to be generated.
x	Causes multiplication problems to be generated.
/	Causes division problems to be generated.
range	Indicates a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. The default *range* is 10.

DESCRIPTION

`arithmetic` types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back `Right!`, and a new problem. If the answer is wrong, it replies `What?`, and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt.

The options, `+`, `-`, `x`, or `/`, determine the kind of problem to be generated. One or more options can be given. If more than one is given, the different types of problems will be mixed in random order. The default is `+-`.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts. For almost all users, the relevant statistic should be time per problem, not percent correct.

FILES

`/usr/games/arithmetic`
Executable file

NAME

autorobots — plays the game of autorobots

SYNOPSIS

autorobots

DESCRIPTION

The object of the game autorobots is to move around inside of the box on the screen without getting eaten by the robots chasing you and without running into any robots or junk heaps. The robots move continuously.

If a robot runs into another robot or junk heap while chasing you, they crash and leave a junk heap.

You start out with 10 robots worth 10 points each. If you defeat all of them, you get 20 robots worth 20 points each. Then 30, and so forth, until you get eaten!

The game keeps track of the top 10 scores and prints them at the end of the game.

The valid commands are described on the screen.

FILES

/usr/games/autorobots
    Executable file

NAME

`back` — plays the game of backgammon

SYNOPSIS

`back`

DESCRIPTION

`back` is a program which provides a partner for the game of backgammon. It is designed to play at three different levels of skill, one of which you must select. In addition to selecting the opponent's level, you may also indicate that you would like to roll your own dice during your turns (for the superstitious players). You will also be given the opportunity to move first. The practice of each player rolling one die for the first move is not incorporated.

The points are numbered 1-24, with 1 being white's extreme inner table, 24 being brown's inner table, 0 being the bar for removed white pieces and 25 the bar for brown. For details on how moves are expressed, type `y` when `back` asks `Instructions?` at the beginning of the game. When `back` first asks `Move?`, type `?` to see a list of move options other than entering your numerical move.

When the game is finished, `back` will ask you if you want postmortem statistics. If you respond with `y`, `back` will attempt to append to or create a file `.backlog` in your home directory.

LIMITATIONS

The only level really worth playing is `expert`, and it only plays the forward game.

Doubling is not implemented.

FILES

`/usr/games/back`
Executable file

`/usr/games/lib/backrules`
File containing rules for `back`

`$HOME/.backlog`
Statistics file

NAME

`bcd` — simulates a punched card corresponding to a text argument

SYNOPSIS

`bcd` *text*

ARGUMENTS

text Specifies data that is input by the user.

DESCRIPTION

`bcd` converts the literal *text* into a form familiar to old-timers. This program works best on hard copy terminals.

FILES

`/usr/games/bcd`
Executable file

NAME

`bj` — plays the game of black jack

SYNOPSIS

`bj`

DESCRIPTION

`bj` is a serious attempt at simulating the dealer in the game of black jack (or twenty-one) as might be found in Reno. The following rules apply:

The bet is $2 every hand.

> You `natural` (black jack) pays $3; a dealer natural loses $2. Both dealer and player naturals is a `push` (no money exchange).
>
> If the dealer has an ace up, you are allowed to make an `insurance` bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where you win $2 if the dealer has a natural, and you lose $1 if the dealer does not.
>
> If you are dealt two cards of the same value, you are allowed to `double`. You are allowed to play two hands, each with one of these cards. (The bet is doubled also; $2 on each hand.)
>
> If a dealt hand has a total of ten or eleven, you may `double down`. You may double the bet ($2 to $4) and receive exactly one more card on that hand.
>
> Under normal play, you may `hit` (draw a card) as long as your total is not over twenty-one. If you `bust` (go over twenty-one), the dealer wins the bet.
>
> When you `stand` (decide not to hit), the dealer hits until a total of seventeen or more has been attained. If the dealer busts, you win the bet.
>
> If both you and the dealer stand, the one with the largest total wins. A tie is a `push`.

The machine deals and keeps score. The following questions will be asked at appropriate times. Each question is answered by `y` followed by a newline for `yes`, or just newline for `no`.

> `?` (means, `do you want a hit?`)
> `Insurance?`
> `Double down?`

Every time the deck is shuffled, the dealer so states and the `action` (total bet) and `standing` (total won or lost) is printed.

To exit, hit the interrupt key and the action and standing will be printed.

FILES

`/usr/games/bj`
    Executable file

NAME

`chase` — plays the game of chase

SYNOPSIS

`chase` [*nrobots*] [*nfences*]

ARGUMENTS

nrobots	Specifies the number of robots to chase you.
nfences	Specifies the number of fences to be displayed.

DESCRIPTION

The object of the game `chase` is to move around inside of the box on the screen without getting eaten by the robots chasing you and without running into anything.

If a robot runs into another robot while chasing you, they crash and leave a junk heap. If a robot runs into a fence, it is destroyed.

If you can survive until all the robots are destroyed, you have won!

If you do not specify either *nrobots* or *nfences,* chase will prompt you for them.

The valid commands are described on the screen.

FILES

`/usr/games/chase`
Executable file

NAME

`craps` — plays the game of craps

SYNOPSIS

`craps`

DESCRIPTION

`craps` is a form of the game of craps that is played in Las Vegas. The program simulates the `roller`, while you (the `player`) place bets. You may choose, at any time, to bet with the roller or with the `House`. A bet of a negative amount is taken as a bet with the House; any other bet is a bet with the roller.

You start off with a bankroll of $2,000. The program prompts with:

```
bet?
```

The bet can be all or part of your bankroll. Any bet over the total bankroll is rejected and the program prompts with `bet?` until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (the player wins or loses depending on whether the bet is placed with the roller or with the House; the odds are even). The ''first'' roll is the roll immediately following a bet:

1. On the first roll:

7 or 11	wins for the roller;
2, 3, or 12	wins for the House;

any other number is the point, roll again (Rule 2 applies).

2. On subsequent rolls:

point	roller wins;
7	House wins;
any other number	roll again.

If you lose the entire bankroll, the House will offer to lend you an additional $2,000. The program will prompt:

```
marker?
```

A `yes` (or `y`) consummates the loan. Any other reply terminates the game.

If you owe the House money, the House reminds you, before a bet is placed, of how many markers are outstanding.

If, at any time, you have a bankroll exceeding $2,000 and an outstanding marker, the House asks:

```
Repay marker?
```

A reply of `yes` (or `y`) indicates your willingness to repay the loan. If only 1 marker is outstanding, it is immediately repaid. However, if more than 1 marker is outstanding, the House asks:

```
How many?
```

markers you would like to repay. If an invalid number is entered (or just a carriage return), an appropriate message is printed and the program will prompt with `How many?` until a valid number is entered.

If you accumulate 10 markers (a total of $20,000 borrowed from the House), the program informs you of the situation and exits.

Should your bankroll exceed $50,000 and you have outstanding markers, the total amount of money borrowed will automatically be repaid to the House.

If you accumulate $100,000 or more, you will break the bank. The program then prompts:

```
New game?
```

to give the House a chance to win back its money.

Any reply other than `yes` is considered to be a `no` (except in the case of `bet?` or `How many?`).

To exit, send an interrupt. The program will indicate whether you won, lost, or broke even.

NOTES

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.

FILES

`/usr/games/craps`
Executable file

NAME

`cribbage` — plays the game of cribbage

SYNOPSIS

`cribbage` [`-e`] [`-q`] [`-r`] *name...*

ARGUMENTS

`-e`	Provides an explanation of the correct score when you make a mistake scoring your hand or crib. (This is especially useful for beginning players.)
name	Specifies the given *name.*
`-q`	Prints a shorter form of all messages; this is only recommended for users who have played the game without specifying this option.
`-r`	Cuts the deck randomly, instead of asking you to cut the deck.

DESCRIPTION

`cribbage` plays the card game cribbage, with the program playing one hand and you the other. The program will initially ask you if the rules of the game are needed; if so, it will print out the appropriate section from *According to Hoyle* with `more`.

The `cribbage` game first asks you whether you wish to play a short game (''once around,'' to 61) or a long game (''twice around,'' to 121). A response of `s` will result in a short game, any other response will play a long game.

At the start of the first game, the program asks you to cut the deck to determine who gets the first crib. You should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. Whoever cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, the program first prints your hand, whose crib it is, and then asks you to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, the program cuts the deck (if it is your crib) or asks you to cut the deck (for its crib); in the latter case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the nondealer (the person who doesn't have the crib) leading the first card. Play continues, as per cribbage, until all cards are exhausted. The program keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. The program requests you to score his hand (and the crib, if it is yours) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

Cards are specified as rank followed by suit. The ranks may be specified as one of: `a`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `t`, `j`, `q`, and `k`, or alternatively, one of: `ace`, `two`, `three`, `four`, `five`, `six`, `seven`, `eight`, `nine`, `ten`, `jack`, `queen`, and `king`, respectively. Suits may be specified as: `s`, `h`, `d`, and `c`, or alternatively as: `spades`, `hearts`, `diamonds`, and `clubs`, respectively. A card may be specified as: *<rank>* *<suit>*, or: *<rank>* of *<suit>*. If the single letter rank and suit designations are used, the space separating the suit and rank may be left out. Also, if only one card of the desired rank is playable, typing the rank is sufficient. For example, if your hand was ''2H, 4D, 5C, 6H, JC, KD'' and it was desired to discard the King of Diamonds, any of the following could be typed:

```
k
king
kd
k d
k of d
king d
king of d
k diamonds
k of diamonds
king diamonds
king of diamonds
```

FILES

`/usr/games/cribbage`
Executable file

See `ttt`(6)

NAME

`fish` — plays the game of ‘‘Go Fish’’

SYNOPSIS

`fish`

DESCRIPTION

`fish` plays the children’s card game of ‘‘Go Fish.’’ The object is to accumulate ‘‘books’’ of 4 cards with the same face value.

The players alternate turns; each turn begins with one player selecting a card from his hand and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, those cards are given to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player’s hand: the second player replies `GO FISH`! The first player then draws a card from the ‘‘pool’’ of undealt cards. If this is the card that was last requested, the player may draw again.

When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing `a`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `10`, `j`, `q`, or `k` when asked.

Pressing RETURN gives you information about the size of my hand and the pool, and tells you about my books. Saying `p` as a first guess puts you into ‘‘pro’’ level; the default is pretty dumb.

FILES

`/usr/games/fish`
  Executable file

NAME

fortune — plays the game of fortune telling

SYNOPSIS

fortune

DESCRIPTION

fortune prints out a random, hopefully interesting, adage.

FILES

/usr/games/fortune
: Executable file

/usr/games/lib/fortunes
: File containing fortunes

NAME

`hangman` — plays the game of hangman

SYNOPSIS

`hangman` [*dictionary*]

ARGUMENTS

dictionary

Specifies an alternate dictionary to be used with the game.

DESCRIPTION

The object of the game `hangman` is to guess the word that the system picks by guessing the letters within that word. The word must be at least seven letters long and be listed in a dictionary. You guess the letters one at a time.

LIMITATIONS

Hyphenated compounds are run together.

FILES

`/usr/games/hangman`

Executable file

`/usr/lib/w2006`

Temporary file

NAME

`life` — plays the game of life

SYNOPSIS

`life [-r]`

ARGUMENTS

`-r` Treats the screen as a torus; that is, the top and bottom lines and the left and right columns are considered adjacent.

DESCRIPTION

`life` is a pattern-generating game set up for interactive use on a video terminal. To play the game, you use a series of commands to set up a pattern on the screen and then let it generate further patterns from that pattern.

For each square in the matrix, look at it and its 8 adjacent neighbors. If the present square is unoccupied and exactly 3 of its neighboring squares are occupied, then that square will be occupied in the next pattern. If the present square is occupied and 2 or 3 of its neighboring squares are occupied, then that square will be occupied in the next pattern. Otherwise, the present square will not be occupied in the next pattern.

The edges of the screen are normally treated as an unoccupied void.

The pattern generation number and the number of occupied squares are displayed in the lower left corner of the screen.

Following is a list of commands available to the user. In these descriptions, *m* and *n* may be replaced by any numbers.

m,*n*`a`
: Adds a block of elements. The first number specifies the horizontal width and the second number specifies the vertical height. If a number is not specified, the default is 1.

n`c` Steps through the next *n* patterns. If no number is specified, step forever. The operation can be cancelled by typing an interrupt.

m,*n*`d`
: Deletes a block of elements. The first number specifies the horizontal width and the second number specifies the vertical height. If a number is not specified, the default is 1.

n`f` Generates a little flier at the present location. The number (modulo 8) determines the direction.

m,*n*`g`
: Moves to absolute screen location. The first number specifies the horizontal location and the second number specifies the vertical location. If a number is not specified, the default is 0.

n`h` Moves left *n* steps. If no number is specified, the default is 1.

n`j` Moves down *n* steps. The default is 1.

n`k` Moves up *n* steps. The default is 1.

n`l` Moves right *n* steps. The default is 1.

n`n` Steps through the next *n* patterns. If no number is specified, generate the next pattern. The operation can be cancelled by typing an interrupt.

`p` Puts the last yanked or deleted block at the present location.

`q` Quits the game.

m`,`*n*`y`
Yanks a block of elements. The first number specifies the horizontal width and the second number specifies the vertical height. If a number is not specified, the default is 1.

`C` Clears the pattern.

n`F` Generates a big flier at the present location. The number (modulo 8) determines the direction.

n`H` Moves to the left margin.

n`J` Moves to the bottom margin.

n`K` Moves to the top margin.

n`L` Moves to the right margin.

*n*CONTROL-H
Moves left *n* steps. If no number is specified, the default is 1.

*n*CONTROL-J
Moves down *n* steps. The default is 1.

*n*CONTROL-K
Moves up *n* steps. The default is 1.

*n*CONTROL-L
Moves right *n* steps. The default is 1.

CONTROL-R
Redraws the screen. This is used for those occasions when the terminal screws up.

`.` Repeats the last add (`a`) or delete (`d`) operation.

`;` Repeats the last move (`h`, `j`, `k`, `l`) operation.

LIMITATIONS

The following features are planned but not implemented.

m,*n*S	Saves the selected area in a file.
R	Restores from a file.
m	Generates a macro command.
!	Escapes the shell.
e	Edits a file.
i	Inputs commands from a file.

FILES

`/usr/games/life`
: Executable file

NAME

`mastermind` — plays the game of Mastermind

SYNOPSIS

`mastermind`

DESCRIPTION

`mastermind` plays the board game Mastermind. The playing field is a number of slots, in which a number of colored pegs can be placed.

The object of the game is to guess a hidden sequence of colored pegs. Each guess consists of a possible sequence of colored pegs. The guesser's opponent then answers with two numbers: the number of pegs in the guess that exactly match the corresponding pegs in the configuration, and the number of pegs in the guess that match in color but not in position. Play continues until the sequence is guessed correctly. Then players change positions and the program will try to guess your hidden sequence.

The guesser's opponent gets one point for each guess made. You have a chance to decide before starting how many slots and how many colors you want to use. Any time it is your turn to enter a guess, you can review the board by typing `review` instead of your guess.

FILES

`/usr/games/mastermind`
Executable file

`/usr/games/mmhow`
Instruction file

NAME

`maze` — generates a maze

SYNOPSIS

`maze`

DESCRIPTION

`maze` asks a few questions and then prints a maze.

LIMITATIONS

Some mazes (especially small ones) have no solutions.

FILES

`/usr/games/maze`
: Executable file

NAME

moo — plays the game of moo

SYNOPSIS

moo

DESCRIPTION

moo is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. You guess four distinct digits being scored on each guess.

A cow is a correct digit in an incorrect position. A bull is a correct digit in a correct position. The game continues until you guess the number (a score of four bulls).

FILES

/usr/games/moo
    Executable file

NAME

number — converts Arabic numerals to English

SYNOPSIS

number

DESCRIPTION

number copies the standard input to the standard output, changing each decimal number to a fully spelled out version.

FILES

/usr/games/number
Executable file

NAME

`quiz` — gives associative knowledge tests on various subjects

SYNOPSIS

`quiz` [`-i`*file*] [`-t`] [*category1 category2*]

ARGUMENTS

category1 category2

Causes `quiz` to ask items chosen from *category1* and expects answers from *category2*, or vice versa. If no categories are specified, `quiz` gives instructions and lists the available categories.

`-i`*file*

Causes the named *file* to be substituted for the default index file. The lines of these files have the syntax:

```
line = category newline | category : line
category = alternate | category | alternate
alternate = empty | alternate primary
primary = character | [category [|option
option = {category}
```

The first category on each line of an index names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax.

`-t` Specifies a `tutorial` mode, where missed questions are repeated later, and material is gradually introduced as you learn.

DESCRIPTION

`quiz` tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, `quiz` reports a score and terminates.

A backslash \ is used, as with `sh`, to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, `quiz` will refrain from asking it.

LIMITATIONS

The construct

```
a | ab
```

doesn't work in an information file. Use `a{b}`.

FILES

`/usr/games/quiz`
Executable file

`/usr/games/lib/quiz/index`
Index file

`/usr/games/lib/quiz/*`
Files containing quiz information

NAME

`rain` — animates raindrops

SYNOPSIS

`rain`

DESCRIPTION

The display of `rain` is modeled after the VAX/VMS program of the same name. The terminal has to be set for 9600 baud to obtain the proper effect.

As with all programs that use `termcap`, the `TERM` environment variable must be set (and exported) to the type of the terminal being used.

FILES

`/usr/games/rain`
: Executable file

`/etc/termcap`
: Terminal capabilities file

NAME

`robots` — plays the game of robots

SYNOPSIS

`robots`

DESCRIPTION

The object of the game `robots` is to move around inside of the box on the screen without getting eaten by the robots chasing you and without running into anything.

If a robot runs into another robot while chasing you, they crash and leave a junk heap.

You start out with 10 robots worth 10 points each. If you defeat all of them, you get 20 robots worth 20 points each. Then 30, etc. Until you get eaten!

The game keeps track of the top 10 scores and prints them at the end of the game.

The valid commands are described on the screen.

FILES

`/usr/games/robots`
Executable file

NAME

`trek` — plays the game of trek

SYNOPSIS

`trek` [[`-a`] *file*]

ARGUMENTS

`-a` Appends the log of the game to the specified *file*; does not truncate.

file Specifies the file onto which the log of the game is written.

DESCRIPTION

`trek` is a game of space glory and war. Below is a summary of commands. For complete documentation, see *Trek* by Eric Allman.

The game will ask you what length game you would like. Valid responses are `short`, `medium`, and `long`. You may also type `restart`, which restarts a previously saved game. You will then be prompted for the skill, to which you must respond `novice`, `fair`, `good`, `expert`, `commodore`, or `impossible`. You should normally start out with `novice` and work up.

In general, throughout the game, if you forget what is appropriate the game will tell you what it expects if you just type in a question mark.

Commands

abandon	capture
cloak up/down	
computer request; ...	damages
destruct	dock
help	impulse course distance
lrscan	move course distance
phasers automatic amount	
phasers manual amt1 course1 spread1 ...	
torpedo course [yes] angle/no	
ram course distance	rest time
shell	shields up/down
srscan [yes/no]	
status	terminate yes/no
undock	visual course
warp warp_factor	

FILES

`/usr/games/trek`
Executable file

NAME

`ttt, cubic` — play the game of tic-tac-toe

SYNOPSIS

```
ttt
```

```
cubic
```

DESCRIPTION

`ttt` is the X and O game popular in the first grade. This is a learning program that never makes the same mistake twice. Although it learns, it learns slowly. It must lose nearly 80 games to completely know the game.

`cubic` plays three-dimensional tic-tac-toe on a 4×4×4 board. Moves are specified as a sequence of three coordinate numbers in the range 1-4.

FILES

`/usr/games/ttt`
    Executable file
`/usr/games/ttt.k`
    File
`/usr/games/cubic`
    Executable file

NAME

`twinkle` — plays the game of twinkle, twinkle little stars

SYNOPSIS

`twinkle` [`-`] [+] [`s` *file*] [*density1* [*density2*]]

ARGUMENTS

- Prints out the present screen density (the percentage of the screen that will be filled with stars) in the lower-left corner of the screen. This number changes as stars go on and off.

+ Does not ''randomize'' before starting. The screen starts out completely blank and stars are added, bit by bit. In this case, the density rises beyond the specified density, then falls to the required percentage.

density1 density2
Specifies the density of the screen. If this option is not specified, density is .5 (50% of the screen is filled with stars).
If only *density1* is given, density is 1/*density1*.
If both *density1* and *density2* are given, density is the resultant of *density1*/(*density1*+*density2*).

`s` *file*
Saves binary density on *file* in case you want to see the density curve that a particular density specification produced during the life of the show.

DESCRIPTION

`twinkle` causes a specified density of ''stars'' to twinkle on the screen.

EXAMPLES

The command:

```
twinkle -+ 2 6
```

would start from a blank screen and twinkle stars to a final density of 2/8, or 25%. The densities would be shown in the lower-left corner, as a three-place decimal.

FILES

`/usr/games/twinkle`
Executable file

NAME

worm — plays the game of growing worm

SYNOPSIS

worm [*size*]

ARGUMENTS

size Specifies the initial length of the worm. Replace *size* with a number between 1 and 75.

DESCRIPTION

In the game worm, you are a little worm, your body is the string of os on the screen, and your head is the @. You move around with the h, j, k, and l keys. If you don't press any keys, you continue in the direction you last moved. The uppercase H, J, K, and L keys move you as if you had pressed the corresponding lowercase key several times (9 for HL and 5 for JK) unless you run into a digit, then it stops.

On the screen you will see a digit. If the worm eats the digit, it will grow longer. The actual amount by which the worm will grow longer depends upon which digit was eaten. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into either the sides of the screen or itself. The current score (how much the worm has grown) is kept in the upper left corner of the screen.

LIMITATIONS

If the initial length of the worm is set to less than one or more than 75, various strange things happen.

FILES

/usr/games/worm
Executable file

NAME

`worms` — plays the game of worms

SYNOPSIS

`worms` [`-field`] [`-length` *n*] [`-number` *n*] [`-trail`]

ARGUMENTS

`-field`
: Fills the entire window (`field`) with the words `WORM`. The characters in the field disappear when the worms cross their path. The worms continue to be active after they have eaten the field.

`-length` *n*
: Specifies the length of the worms to be *n* characters long.

`-number` *n*
: Creates *n* number of worms in the field.

`-trail`
: Causes each worm to leave a trail of dots (`.`) behind it.

DESCRIPTION

`worms` first clears the active window, then simulates two worms made of of asterisks (`*`), and one worm made of dollar signs (`$`) wiggling around the window area, or `field`. Each worm is one character in width and approximately 20 characters in length.

The worms will continue in motion until an interrupt is sent.

STATUS MESSAGES AND VALUES

`Invalid length`
: Value not in range 2 <= length <= 1024

`Invalid number of worms`
: Value not in range 1 <= number <= 40

`TERM: parameter not set`
: The `TERM` environment variable is not defined. To fix things, run the commands

    `TERM=`*terminal-type*
    `export TERM`

`Unknown terminal type`
: The terminal type (as determined from the `TERM` environment variable) is not defined in `/etc/termcap`.

`Terminal not capable of cursor motion`
: The terminal is incapable of running this program.

LIMITATIONS

The lower right character position will not be properly updated on a terminal that wraps at the right margin.

Terminal initialization is not performed.

FILES

`/usr/games/worms`
: Executable file

`/etc/termcap`
: Terminal capabilities file

NAME

wump — plays the game of hunt-the-wumpus

SYNOPSIS

`wump`

DESCRIPTION

`wump` invokes the game of ''Hunt the Wumpus.'' A Wumpus is a creature that lives in a cave containing several rooms that are connected by tunnels.

You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile trying to avoid being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats, which are likely to pick you up and drop you into some random room.

The program asks various questions which you answer one per line. You can ask the program to give you a more detailed description of the questions, if you wish.

LIMITATIONS

It will never replace Adventure.

NOTES

This program is based on one described in ''People's Computer Company,'' 2, 2 (November 1973).

FILES

`/usr/games/wump`
Executable file