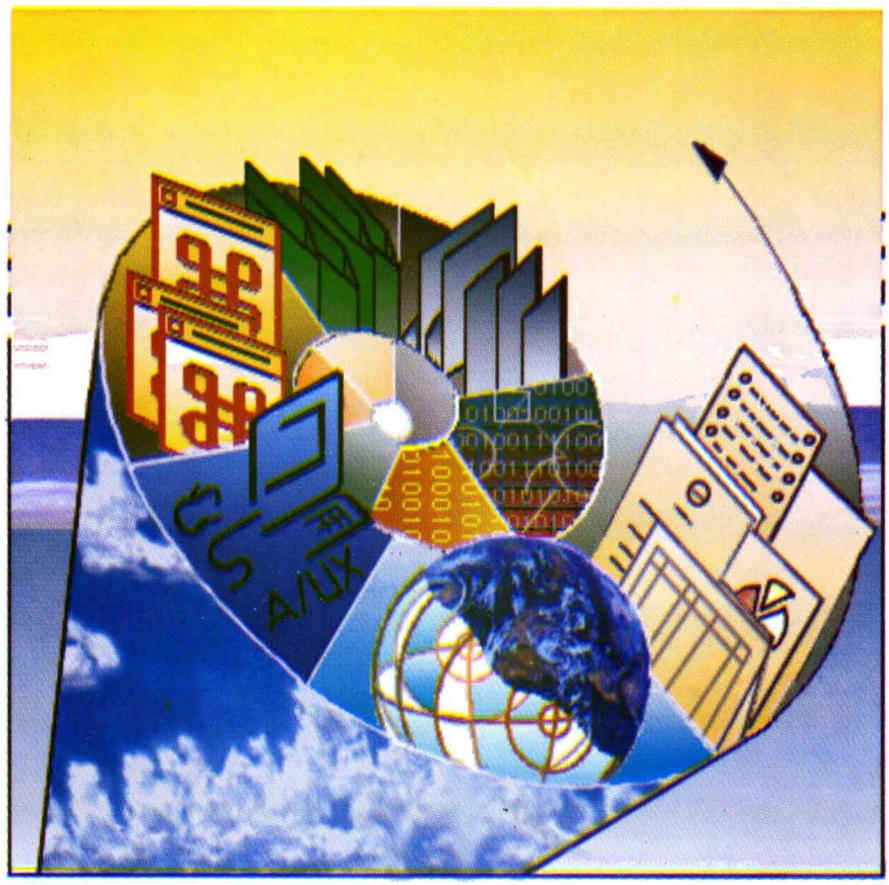
A/UX® Command Reference
Sections 1(M-Z) and 6



A/UX® Command Reference

Sections 1(M-Z) and 6

030-0782

APPLE COMPUTER, INC.

Apple Computer, Inc.
20525 Mariani Ave.
Cupertino, California 95014
(408) 996-1010

Simultaneously published in the United States and Canada.

030-0782

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or in the media on which a software product is distributed, Apple will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to Apple or an authorized Apple dealer during the 90-day period after you purchased the software. In addition, Apple will replace damaged software media and manuals for as long as the software product is included in Apple's Media Exchange Program. While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apple dealer for program coverage and details. In some countries the replacement period may be different, check with your authorized Apple dealer.

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

A/UX Command Reference

Contents

Preface

Conventions Used in This Manual

A/UX® manuals follow certain conventions regarding presentation of information. Words or terms that require special emphasis appear in specific fonts within the text of the manual. The following sections explain the conventions used in this manual.

Significant fonts

Words that you see on the screen or that you must type exactly as shown appear in `Courier` font. For example, when you begin an A/UX work session, you see the following on the screen:

```
login:
```

The text shows `login:` in `Courier` typeface to indicate that it appears on the screen. If the next step in the manual is

Enter `start`

`start` appears in `Courier` to indicate that you must type in the word. Words that you must replace with a value appropriate to a particular set of circumstances appear in *italics*. Using the example just described, if the next step in the manual is

`login:` *username*

you type in your name—`Laura`, for example— so the screen shows:

```
login: Laura
```

Key presses

Certain keys are identified with names on the keyboard. These modifier and character keys perform functions, often in combination with other keys. In the manuals, the names of these keys appear in the format of an Initial Capital letter followed by SMALL CAPITAL letters.

The list that follows provides the most common keynames.

RETURN	DELETE	SHIFT	ESCAPE
OPTION	CAPS LOCK	CONTROL	

For example, if you enter

```
Applee
```

instead of

```
Apple
```

you would position the cursor to the right of the word and press the DELETE key once to erase the additional *e*.

For cases in which you use two or more keys together to perform a specific function, the keynames are shown connected with hyphens. For example, if you see

Press CONTROL-C

you must press CONTROL and C simultaneously (CONTROL-C normally cancels the execution of the current command).

Terminology

In A/UX manuals, a certain term can represent a specific set of actions. For example, the word *Enter* indicates that you type in an entry and press the RETURN key. If you were to see

Enter the following command: `whoami`

you would type `whoami` and press the RETURN key. The system would then respond by identifying your login name.

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

Term	Action
Enter	Type in the entry and press the RETURN key
Press	Press a *single* letter or key *without* pressing the RETURN key
Type	Type in the letter or letters *without* pressing the RETURN key
Click	Press and then immediately release the mouse button

Term	Action
Select	Position the pointer on an item and click the mouse button
Drag	Position the pointer on an icon, press and hold down the mouse button while moving the mouse. Release the mouse button when you reach the desired position.
Choose	Activate a command title in the menu bar. While holding down the mouse button, drag the pointer to a command name in the menu and then release the mouse button. An example is to drag the File menu down until the command name Open appears highlighted and then release the mouse button.

Syntax notation

A/UX commands follow a specific order of entry. A typical A/UX command has this form:

`command` [*flag-option*] [*argument*] . . .

The elements of a command have the following meanings.

Element	Description
`command`	Is the command name.
flag-option	Is one or more optional arguments that modify the command. Most flag-options have the form [-opt...] where opt is a letter representing an option. Commands can take one or more options.
argument	Is a modification or specification of the command; usually a filename or symbols representing one or more filenames.

Element	Description
brackets ([ ])	Surround an optional item—that is, an item that you do not need to include for the command to execute.
ellipses (...)	Follow an argument that may be repeated any number of times.

For example, the command to list the contents of a directory (`ls`) is followed below by its possible flag options and the optional argument *names*.

```
ls [-R] [-a] [-d] [-C] [-x] [-m] [-l] [-L]
[-n] [-o] [-g] [-r] [-t] [-u] [-c] [-p] [-F]
[-b] [-q] [-i] [-s] [names]
```

You can enter

```
ls -a /users
```

to list all entries of the directory `/users`, where

`ls`	Represents the command name
`-a`	Indicates that *all* entries of the directory be listed
`/users`	Names which directory is to be listed

Command reference notation

Reference material is organized by section numbers. The standard A/UX cross-reference notation is

cmd(sect)

where *cmd* is the name of the command, file, or other facility; *sect* is the section number where the entry resides.

- ☐ Commands followed by section numbers (1M), (7), or (8) are listed in *A/UX System Adminstrator's Reference*.
- ☐ Commands followed by section numbers (1), (1C), (1G), (1N), and (6) are listed in *A/UX Command Reference*.
- ☐ Commands followed by section numbers (2), (3), (4), and (5) are listed in *A/UX Programmer's Reference*.

For example,

```
cat(1)
```

refers to the command `cat`, which is described in Section 1 of *A/UX Command Reference*. References can also be called up on the screen. The `man` command or the `apropos` command displays pages from the reference manuals directly on the screen. For example, enter the command

```
man cat
```

In this example, the manual page for the `cat` command including its description, syntax, options, and other pertinent information appears on the screen. To exit, continue pressing the space bar until you see a command prompt, or press Q at any time to return immediately to your command prompt. The manuals often refer to information discussed in another guide in the suite. The format for this type of cross reference is "Chapter Title," *Name of Guide*. For a complete description of A/UX guides, see *Road Map to A/UX Documentation*. This guide contains descriptions of each A/UX guide, the part numbers, and the ordering information for all the guides in the A/UX documentation suite.

Introduction

to the A/UX Reference Manuals

1. How to use the reference manuals

A/UX Command Reference, *A/UX Programmer's Reference*, and *A/UX System Administrator's Reference* are reference manuals for all the programs, utilities, and standard file formats included with your A/UX® system.

The reference manuals constitute a compact encyclopedia of A/UX information. They are not intended to be tutorials or learning guides. If you are new to A/UX or are unfamiliar with a specific functional area (such as the shells or the text formatting programs), you should first read *A/UX Essentials* and the other A/UX user guides. After you have started worked with A/UX, the reference manuals help you understand new features or refresh your memory about command features you already know.

2. Information contained in the reference manuals

A/UX reference manuals are divided into three volumes:

- The two-part *A/UX Command Reference* contains information for the general user. It describes commands you type at the A/UX prompt that list your files, compile programs, format text, change your shell, and so on. It also includes programs used in scripts and command language procedures. The commands in this manual generally reside in the directories `/bin`, `/usr/bin` and `/usr/ucb`.
- The two-part *A/UX Programmer's Reference* contains information for the programmer. It describes utilities for programming, such as system calls, file formats of subroutines, and miscellaneous programming facilities.
- *A/UX System Administrator's Reference* contains information for the system administrator. It describes commands you type at the A/UX prompt to control your machine, such as accounting

commands, backing up your system, and charting your system's activity. These commands generally reside in the directories `/etc`, `/usr/etc`, and `/usr/lib`.

These areas can overlap. For example, if you are the only person using your machine, then you are both the general user and the system administrator.

To help direct you to the correct manual, you may refer to *A/UX Reference Summary and Index*, which is a separate volume. This manual summarizes information contained in the other A/UX reference manuals. The three parts of this manual are a classification of commands by function, a listing of command synopses, and an index.

3. How the reference manuals are organized

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

Manual pages are collated alphabetically by the primary name associated with each. For the individual sections, a table of contents is provided to show the sequence of manual pages. A notable exception to the alphabetical sequence of manual pages is the first entry at the start of each section. As a representative example, `intro.1` appears at the start of Section 1. These `intro.`*section-number* manual pages are brought to the front of each section because they introduce the

other man pages in the same section, rather than describe a command or similar provision of A/UX.

Each of the reference manuals includes at least one complete section of man pages. For example, the *A/UX Command Reference* contains sections 1 and 6. However, since Section 1 (User Commands) is so large, this manual is divided into two volumes, the first containing Section 1 commands that begin with letters A through L, and the second containing Section 6 commands and Section 1 commands that begin with letters M through Z. The sections included in each volume are as follows.

A/UX Command Reference contains sections 1 and 6. Note that both of these sections describe commands and programs available to the general user.

- Section 1—User Commands
  The commands in Section 1 may also belong to a special category. Where applicable, these categories are indicated by the letter designation that follows the section number. For example, the N in `ypcat`(1N) indicates networking as described following.

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

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

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

- Section 6—User Commands
  This section contains all the games, such as `cribbage` and `worms`.

A/UX Programmer's Reference contains sections 2 through 5.

- 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 networking as described following.

  2N Networking system calls

  2P POSIX system calls

- Section 3—Subroutines
  This section describes the available subroutines. The binary versions are in the system libraries in the `/lib` and `/usr/lib` directories. The section includes six 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 networking as described following.

  3C C and assembler 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. Where applicable, these categories are indicated by the letter designation that follows the section number. For example, the N in `protocols`(4N) indicates networking as described following.

4N Networking formats

- Section 5—Miscellaneous facilities
  This section contains various character sets, macro packages, and other miscellaneous formats. 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 as described following. by the letter designation in parenthesis at the top of the page:

  5F Protocol families

  5P Protocol descriptions

A/UX System Administrator's Reference contains sections 1M, 7 and 8.

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

- Section 7—Drivers and Interfaces for Devices
  This section discusses the drivers and interfaces through which devices are normally accessed. While access to one or more disk devices is fairly transparent when you are working with files, the provision of *device files* permits you more explicit modes with which to access particular disks or disk partitions, as well as other types of devices such as tape drives and modems. For example, a tape device may be accessed in automatic-rewind mode through one or more of the device file names in the `/dev/rmt` directory (see `tc`(7)). The **FILES** sections of these manual pages identify all the device files supplied with the system as well as those that are automatically generated by certain A/UX configuration utilties. The names of the man pages generally refer to device names or device driver names, rather than the names of the device files themselves.

- Section 8—A/UX Startup Shell Commands
  This section describes the commands that are available from within the A/UX Startup Shell, including detailed descriptions of those that contribute to the boot process and those that help with

the maintenance of file systems.

4. How a manual entry is organized

Each section of the reference volumes has an introductory man page, such as `intro`(1), followed by the remaining man page entries arranged alphabetically. Each entry is numbered separately (that is, each entry begins on a page numbered "1").

The entry name normally appears twice, once in each upper corner of a page. Like dictionary guide words, these names appear at the top of every physical page. After each name is the section number and, if applicable, a category letter enclosed in parenthesis, such as (1) or (2N).

Some entries describe several routines or commands. These multiple subentries are listed under the main entry, and each subentry refers back to the main entry. For example, `chown` and `chgrp` share a page with the 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 only included in *A/UX Command Reference* and *A/UX System Administrator's Reference*.)

All of the entries have a common format, and may include any of the following parts:

NAME
is the name or names and a brief description.

SYNOPSIS
describes the syntax for using the command or routine.

DESCRIPTION
discusses what the program does.

FLAG OPTIONS
discusses the flag options.

EXAMPLES
gives an example or examples of usage.

RETURN VALUE
describes the value returned by a function.

ERRORS
describes the possible error conditions.

FILES
lists the filenames that are used by the program.

SEE ALSO
provides pointers to related information.

DIAGNOSTICS
discusses the diagnostic messages that may be produced. Self-explanatory messages are not listed.

WARNINGS
points out potential pitfalls.

BUGS
gives known bugs and sometimes deficiencies. Occasionally, it describes the suggested fix.

5. Locating information in the reference manuals

The directory for the reference manuals, *A/UX Reference Summary and Index*, can help you locate information through its index and summaries. The tables of contents within each of the reference manuals can be used also.

5.1 Table of contents

Each reference manual contains an overall table of contents and individual section contents. The general table of contents lists the overall contents of each volume. The more detailed section contents lists the manual pages contained in each section and a brief description of their function. Note that for the most part entries appear in alphabetic order within each section.

5.2 Commands by function

This summary classifies the A/UX user and administration commands by the general, or most important function they perform. The complete descriptions of these commands are found in *A/UX Command Reference* and *A/UX System Administrator's Reference*. Each is mentioned just once in this listing.

The summary gives you a broader view of the commands that are available and the context in which they are most often used.

5.3 Command synopses

This section gives the syntax requirements of the commands in *A/UX Command Reference* and *A/UX System Administrator's Reference* and is provided as an even shorter reference to help you use commands you are already familiar with.

5.4 Index

The index lists key terms associated with A/UX subroutines and commands. These key terms allow you to locate an entry when you don't know the command or subroutine name.

The key terms were constructed by examining the meaning and usage of the A/UX manual pages. It is designed to be more discriminating and easier to use than the traditional permuted index, which lists nearly all words found in the manual page NAME sections.

The entries are alphabetized by key terms. Each entry consists of a key term, in bold type at the left of the page, followed by a list of one or more alphabetized command or subroutine references. For example, if you wanted to know how to set tabs on your terminal, you might look for the key term *tabs*. You would find the following entry:

tabs
`expand`(1) - expand tabs to spaces, and vice versa
`tabs` - set tabs on a terminal

The list of references contains two lines for command entries whose NAME sections contain the word *tabs*; these are `expand` and `tabs`. Both commands are in Section 1, which is in *A/UX Command Reference*. The command you want is `tabs`, which is in the second volume, *A/UX Command Reference*.

5.5 Online documentation

Besides the paper documentation in the reference manuals, A/UX provides several ways to search and read the contents of each reference from your A/UX system.

To see a manual page displayed on your screen, enter the `man(1)` command followed by the name of the entry you want to see. For example,

```
man passwd
```

To see the description phrase from the NAME section of any manual page, enter the `whatis` command followed by the name of the entry you want to see. For example,

```
whatis apropos
```

To see a list of all manual pages whose descriptions contain a given keyword or string, enter the `apropos` command followed by the word or string. For example,

```
apropos remove
```

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

Table of Contents

Section 1: User Commands (M-Z)

NAME

m4 — macro processor

SYNOPSIS

m4 [−B*int*] [−e] [−H*int*] [−s] [−S*int*] [−T*int*] [−D*name*[=*val*]] [−U*name*] [*file...*]

DESCRIPTION

m4 is a macro processor intended as a front end for C and other languages. Each of the argument files is processed in order. If there are no files or if a filename is -, the standard input is read. The processed text is written on the standard output.

FLAG OPTIONS

The flag options and their effects are as follows:

−B*int* Change the size of the push-back and argument collection buffers from the default of 4096.

−e Operate interactively. Interrupts are ignored and the output is unbuffered.

−H*int* Change the size of the symbol table hash array from the default of 199. The size should be prime.

−s Enable line sync output for the C preprocessor (`#line...`).

−S*int* Change the size of the call stack from the default of 100 slots. Macros take 3 slots, and nonmacro arguments take 1.

−T*int* Change the size of the token buffer from the default of 512 bytes.

To be effective, the flag options noted above must appear before any filenames and before any −D or −U flags:

−D*name*[=*val*]
Define *name* to *val* or to null in the absense of *val*.

−U*name*
Undefine *name*.

Macro calls have the form:

name(*arg1*,*arg2*,...,*argn*)

The right parenthesis, (, must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore

(_), where the first character is not a digit.

Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses that happen to turn up within the value of a nested call are as effective as those in the original input text. After the argument collection, the value of the macro is pushed back onto the input stream and rescanned.

Built-in Macros

`m4` makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

`define` Install the second argument as the value of the macro whose name is the first argument. Each occurrence of `$`*n* in the replacement text, where *n* is a digit, is replaced by the *n*th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; `$#` is replaced by the number of arguments; `$*` is replaced by a list of all the arguments separated by commas; `$@` is like `$*`, but each argument is quoted (with the current quotes).

`undefine` Remove the definition of the macro named in the argument.

`defn` Return the quoted definition of the argument(s). This macros is useful for renaming macros, especially built-in macros.

`pushdef` Similar to `define`, but also save any previous definition.

`popdef` Remove the current definition of the argument(s), exposing the previous one, if any.

`ifdef` If the first argument is defined, install the second argument as its value; otherwise, install the third argu-

ment. If there is no third argument, the value is null. The word `unix` is predefined on the UNIX® system versions of `m4`.

`shift` Return all but the first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that is subsequently performed.

`changequote`
Change quote symbols to the first and second arguments. The symbols may be up to five characters long. `changequote` without arguments restores the original values, that is, `` ` ' ``.

`changecom`
Change left and right comment markers from the default # and newline. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes newline. With two arguments, both markers are affected. Comment markers may be up to five characters long.

`divert` Change the current output stream to its (digit-string) argument. `m4` maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. Output diverted to a stream other than 0 through 9 is discarded.

`undivert` Causes immediate output of text from diversions named as arguments, or all diversions if there is no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

`divnum` Return the value of the current output stream.

`dnl` Read and discard characters up to and including the next newline.

`ifelse` Has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6, and 7. Otherwise, the value is either the fourth string or, if it is not present, null.

`incr`	Return the value of the argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
`decr`	Return the value of the argument decremented by 1.
`eval`	Evaluate the argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, –, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~ as well as relationals and parentheses. Octal and hexidecimal numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
`len`	Return the number of characters in the argument.
`index`	Return the position in the first argument where the second argument begins (zero-origin), or –1 if the second argument does not occur.
`substr`	Return a substring of its first argument. The second argument is a zero-origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
`translit`	Transliterate the characters in the first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
`include`	Return the contents of the file named in the argument.
`sinclude`	Same as `include`, except that nothing is returned if the file is inaccessible.
`syscmd`	Execute the system command given in the first argument. No value is returned.
`sysval`	Is the return code from the last call to `syscmd`.
`maketemp`	Fill in a string of XXXXX in the argument with the current process ID.
`m4exit`	Cause immediate exit from `m4`. Argument 1, if given, is the exit code. The default is 0.

`m4wrap` Push back argument 1 at final EOF. An example is: `m4wrap('cleanup()')`.

`errprint` Print the argument on the diagnostic output file.

`dumpdef` Print current names and definitions for the named items, or for all items if no arguments are given.

`traceon` With no arguments, turn on tracing for all macros, including built-ins. Otherwise, turn on tracing for named macros.

`traceoff` Turn off tracing globally and for any macros specified. Macros specifically traced by `traceon` can be untraced only by specific calls to `traceoff`.

EXAMPLES

```
m4 file1 file2 > outputfile
```

runs the `m4` macro processor on the files `file1` and `file2` and redirects the output into `outputfile`.

FILES

`/usr/bin/m4`

SEE ALSO

cc(1), cpp(1).

"m4 Reference," in *A/UX Programming Languages and Tools, Volume 2.*

See machid(1)

NAME

`m68k, pdp11, u3b, u3b2, u3b5, u3b15, vax` — provide truth value about processor type

SYNOPSIS

```
m68k

pdp11

u3b

u3b2

u3b5

u3b15

vax
```

DESCRIPTION

The following commands (corresponding to *programs*) will return a true value (exit code of 0) if *program* currently runs on a processor that is indicated by the command name.

`m68k`	True if *program* currently runs on a 680x0.
`pdp11`	True if *program* currently runs on a PDP-11/45 or PDP-11/70.
`u3b`	True if *program* currently runs on a 3B 20S.
`u3b2`	True if *program* currently runs on a 3B 2 computer.
`u3b5`	True if *program* currently runs on a 3B 5 computer.
`u3b15`	True if *program* currently runs on a 3B 15 computer.
`vax`	True if *program* currently runs on a VAX-11/750 or VAX-11/780.

The commands that do not apply will return a false (nonzero) value. These commands are often used within `make`(1) *makefiles* and shell procedures to increase portability.

FILES

```
/bin/m68k
/bin/pdp11
/bin/u3b
/bin/u3b2
/bin/u3b5
/bin/u3b15
/bin/vax
```

SEE ALSO

csh(1), ksh(1), make(1), sh(1), test(1), true(1).

NAME

macref — produce cross-reference listing of macro files

SYNOPSIS

macref [-t] [-s] [-n] [--] *file...*

DESCRIPTION

The macref program reads the named files (which are assumed to be nroff(1)/troff(1) input) and produces a cross-reference listing of the symbols in the input.

A -t in the command line causes a macro table of contents to be printed. The -s flag option causes symbol-use statistics to be printed. The -n flag option causes one line to be printed for each reference to a symbol. The options may be grouped behind one -. You may use -- to delimit the end of options. macref does not accept - as standard input.

The default output is a list of the symbols found in the input, each accompanied by a list of all references to that symbol. macref lists the symbols alphabetically in the leftmost column, with the references following to the right. Each reference is given in the form:

[[(*NMname*)] *Mname*—] *type lnum* [#]

where the fields have the following meanings:

Mname the name of the macro within which the reference occurs. This field is missing if the reference occurs outside a macro. Any names listed in the *NMname* part are macros within which *Mname* is defined.

type the type associated, by context, with this occurrence of the symbol. The types may be:

- r request
- m macro
- d diversion
- s string
- n number register
- p parameter (e.g. \$*x* is a parameter reference to *x*. Note that parameters are never modified, and that the only valid parameter symbol names are 1, 2, ... 9).

lnum the line number on which the reference occurred.

this reference modifies the value of the symbol.

Generated names are listed under the artificial symbol name ~sym.

FILES

/usr/bin/macref

SEE ALSO

nroff(1), troff(1).

NAME

mactoiso, isotomac — convert from Macintosh® encoding to International Standards Organization (ISO) encoding

SYNOPSIS

mactoiso [-c *char*] [*file*]
isotomac [-c *char*] [*file*]

DESCRIPTION

mactoiso reads from the optional *file* argument or from standard input, characters encoded according to the Macintosh character set and writes them to standard output converted to the ISO 8859-1 character-set encoding scheme.

isotomac does the same type of conversion except that it reads characters from the ISO character set as input and writes them to standard output converted to the Macintosh character set. Each character set contain characters that are not represented in the other's character set. The command by default places a blank character in place of the characters that are not represented in the other character set. The -c option allows the user to specify the use of another character besides the default blank character.

DIAGNOSTICS

The exit status is 0 if everything is OK and 1 for a usage error.

SEE ALSO

charcvt(3c), mac(5), iso(5).

NAME

mail, rmail — send mail to users or read mail

SYNOPSIS

mail [-e] [-f*file*] [-p] [-q] [-r] [-t] *address* ...

rmail [-t] *address* ...

DESCRIPTION

mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a ?, and a line is read from the standard input to determine the disposition of the message:

newline	Go on to next message.
+	Same as newline.
d	Delete message and go on to next message.
p	Print message again.
-	Go back to previous message.
s [*file*...]	Save message in the named files (mbox is default).
w [*file*...]	Save message, without its header, in the named files (mbox is default).
m [*address*...]	Mail the message to the named *addresses* (yourself is default).
q	Put undeleted mail back in the *mailfile* and stop.
EOT (CONTROL-d)	Same as q.
x	Put all mail back in the *mailfile* unchanged and stop.
!*command*	Escape to the shell to perform *command*.
*	Print a command summary.

The optional arguments alter the printing of the mail:

-e causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.

-p causes all mail to be printed without prompting for disposition.

-q causes mail to terminate after interrupts. Normally an interrupt only causes the termination of the message being printed.

-r causes messages to be printed in first-in, first-out order.

-f*file* causes mail to use *file* (e.g., mbox) instead of the default *mailfile*.

When *address* is named, mail takes the standard input up to an end-of-file (or up to a line consisting of just a .) and adds it to each *address*'s *mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e., From...) are preceded with a >. The -t flag option causes the message to be preceded by all *address*es the mail is sent to. An *address* is usually a user name recognized by login(1). If an *address* being sent mail is not recognized, or if mail is interrupted during input, the file dead.letter will be saved to allow editing and resending. Note that this is regarded as a temporary file in that it is recreated every time needed, erasing the previous contents of dead.letter.

To denote a recipient on a remote system, prefix *address* by the system name and exclamation mark (see uucp(1C)). Everything after the first exclamation mark in *address* is interpreted by the remote system. In particular, if *address* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying a!b!cde as a recipient's name causes the message to be sent to user b!cde on system a. System a will interpret that destination as a request to send the message to user cde on system b. This might be useful, for instance, if the sending system can access system a but not system b, and system a has access to system b. mail will not use uucp if the remote system is the local system name (i.e., localsystem!user).

The *mailfile* may be manipulated in two ways to alter the function of mail. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *address*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *address*. This is especially useful to forward all of a user's mail to one machine in a multiple machine environment. In

order for forwarding to work properly, the *mailfile* should have `mail` as group ID, and the group permission should be read-write.

`rmail` permits only the sending of mail; `uucp`(1C) uses `rmail` as a security precaution.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using `mail`.

EXAMPLES

```
mail cj
```

accepts whatever message is typed up to an EOF. The user `cj` will be notified that he has mail the next time he logs in.

If you want to read mail that has been sent to you, simply type

```
mail
```

FILES

`/bin/mail`	
`/bin/rmail`	
`/etc/passwd`	to identify sender and locate user addresses
`/usr/mail/`*user*	incoming mail for *user*; i.e., the *mailfile*
`$HOME/mbox`	saved mail
`$MAIL`	variable containing pathname of *mailfile*
`/tmp/ma*`	temporary file
`/usr/mail/*.lock`	lock for mail directory
`$HOME/dead.letter`	unmailable text

SEE ALSO

`biff`(1), `login`(1), `mailx`(1), `uucp`(1C), `write`(1).

BUGS

Conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by typing a `p`.

NAME

mailx — interactive message processing system

SYNOPSIS

mailx [-d] [-e] [-f [*filename*]] [-F] [-h *number*] [-H] [-i] [-n] [-N] [-r *address*] [-s *subject*] [-u *user*] [-U] [*name* ...]

DESCRIPTION

mailx provides a flexible environment for sending and receiving messages electronically. When reading mail, mailx provides commands to facilitate saving, deleting, and responding to messages. When sending mail, mailx allows editing, reviewing, and other modifications of the message as it is entered.

Incoming mail is stored in a standard file for each user, called the system *mailbox* for that user, usually named /usr/mail/*name*. (You may alter this default by using the -f flag option, as shown later.) When mailx is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is named mbox and is normally located in the user's home directory (see ENVIRONMENT VARIABLES, later, for a description of this file). Messages remain in this file until forcibly removed.

On the command line, flag options start with a dash (-) and any other arguments are taken to be destinations (recipients). If no recipients are specified, mailx will attempt to read messages from the *mailbox*. Flag options are:

-d	Turn on debugging output; neither particularly interesting nor recommended.
-e	Test for presence of mail. mailx prints nothing and exits with a successful return code if there is mail to read.
-f [*filename*]	Read messages from *filename* instead of *mailbox*. If no *filename* is specified, mbox is used.
-F	Record the message in a file named after the first recipient. Overrides the record variable, if set (see ENVIRONMENT VARIABLES).
-h *number*	The number of network *hops* made so far. This is provided for network software to avoid infinite delivery loops.

-H	Print header summary only.
-i	Ignore interrupts. See also ignore in ENVIRONMENT VARIABLES.
-n	Do not initialize from the system default /usr/lib/mailx/mailx.rc.
-N	Do not print initial header summary.
-r *address*	Pass *address* to network delivery software. All tilde commands are disabled.
-s *subject*	Set the Subject header field to *subject*.
-u *user*	Read *user*'s *mailbox*. This is only effective if *user*'s *mailbox* is not read-protected.
-U	Convert uucp style addresses to internet standards. Overrides the conv environment variable.

To terminate a mail message, type an end-of-file, or enter a single period at the start of a line.

When reading mail, mailx is in command mode. A header summary of the first several messages is displayed, followed by a prompt indicating mailx can accept regular commands (see COMMANDS later in this section). When sending mail, mailx is in input mode. If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, mailx will read the message and store it in a temporary file. Commands may be entered by beginning a line with the escape character (tilde (~) by default) followed by a single command letter and optional arguments. See TILDE ESCAPES later in this section, for a summary of these commands.

At any time, the behavior of mailx is governed by a set of environment variables. These are flags and valued parameters that are set and cleared via the se[t] and uns[et] commands. See ENVIRONMENT VARIABLES, later, for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the

standard input, such as lp(1) for recording outgoing mail on paper. Alias groups are set by the a[lias] command (see COMMANDS, later) and are lists of recipients of any type.

Regular commands are of the form:

[*command*] [*msglist*] [*arguments*]

If no command is specified in command mode, p[rint] is assumed. In input mode, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a "current" message, marked by a > in the header summary. Many commands take an optional list of messages (*msglist*) to operate on, which defaults to the current message. A *msglist* is a list of message specifications separated by spaces, which may include:

n	Message number *n*.
.	The current message.
^	The first undeleted message.
$	The last message.
*	All messages.
n–m	An inclusive range of message numbers.
user	All messages from *user*.
/string	All messages with *string* in the subject line (case ignored).
:c	All messages of type *c*, where *c* is one of d deleted messages n new messages o old messages r read messages u unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. Filenames, where expected, are expanded via the normal shell conventions (see csh(1)). Special characters are recognized by certain commands and are documented with the commands later.

At startup time, mailx reads commands from a system-wide file (/usr/lib/mailx/mailx.rc) to initialize certain parameters, then from a private startup file ($HOME/.mailrc) for personalized variables. Most regular commands are legal inside startup files, the most common use being to set up initial display options and alias lists. The following commands are not legal in the startup file: !, C[opy], e[dit], fo[llowup], F[ollowup], ho[ld], m[ail], pre[serve], r[eply], R[eply], sh[ell], and v[isual]. Any errors in the startup file cause the remaining lines in the file to be ignored.

COMMANDS

The following is a complete list of mailx commands:

!*shell-command* Escape to the shell. See SHELL (ENVIRONMENT VARIABLES).

comment Null command (comment). This may be useful in .mailrc files.

= Prints the current message number.

? Prints a summary of commands.

a[lias] *alias name...*
g[roup] *alias name...*
Declares an a[lias] or g[roup] for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the .mailrc file.

alt[ernates] *name. . .*
Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, alt[ernates] prints the current list of alternate names. (See also allnet under ENVIRONMENT VARIABLES.)

cd [*directory*]
ch[dir] [*directory*]
Changes directory. If *directory* is not

specified, $HOME is used.

c[opy] [*filename*]
c[opy] [*msglist*] Copies messages to the file without marking the messages as saved. Otherwise equivalent to the s[ave] command.

C[opy] [*msglist*] Saves the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the S[ave] command.

d[elete] [*msglist*] Deletes messages from the *mailbox*. If autoprint is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES).

di[scard] [*header-file...*]
ig[nore] [*header-file...*]
Suppresses printing of (discard or ignore) the specified header fields when displaying messages on the screen. Examples of header fields to ignore are status and cc. The fields are included when the message is saved. The P[rint] and T[ype] commands override this command.

dp [*msglist*]
dt [*msglist*] Deletes the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a d[elete] command followed by a p[rint] command.

ec[ho] *string* Echos the given strings (like echo(1)).

e[dit] [*msglist*] Edits the given messages. The messages are placed in a temporary file and the EDITOR variable is used to get the name of the editor (see ENVIRONMENT VARIABLES). Default editor is ed(1).

ex[it]
x[it]
: Exits from mailx, without changing the *mailbox*. No messages are saved in the mbox (see also q[uit]).

fi[le] [*filename*]
fold[er] [*filename*]
: Quits the current file of messages and read in the specified file (folder). Several special characters are recognized when used as filenames, with the following substitutions:

%	the current *mailbox*.
%*user*	the *mailbox* for *user*.
#	the previous file.
&	the current mbox.

Default file is the current *mailbox*.

folders
: Prints the names of the files in the directory set by the folder variable (see ENVIRONMENT VARIABLES).

fo[llowup] [*message*]
: Responds to (follow up on) a message, recording the response in a file whose name is derived from the author of the message. Overrides the record variable, if set. See also the F[ollowup], S[ave], and C[opy] commands and outfolder (ENVIRONMENT VARIABLES).

F[ollowup] [*msglist*]
: Responds to (Follow up on) the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the fo[llowup], S[ave], and C[opy] commands and outfolder (ENVIRONMENT VARIABLES).

f[rom] [*msglist*] Prints the header summary ("from" portion) for the specified messages.

g[roup] *alias name...*
a[lias] *alias name...*
Declare an a[lias] or g[roup] for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the .mailrc file.

h[eaders] [*message*]
Prints the page of headers which includes the message specified. The screen variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the z command.

hel[p] Prints a summary (help list) of commands.

ho[ld] [*msglist*]
pre[serve] [*msglist*]
Holds (Preserves) the specified messages in the *mailbox*.

i[f] *s|r*
mail-commands
el[se]
mail-commands
en[dif]
Conditional execution, where *s* will execute following *mail-commands*, up to an el[se] or en[dif], if the program is in *send* mode (that is, not receiving or reading mail), and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the .mailrc file.

ig[nore] *header-file*
di[scard] *header-file*
Suppresses printing of (ignore or discard) the specified header fields when displaying messages on the screen. Examples of header fields to ignore are status and cc. All

fields are included when the message is saved. The P[rint] and T[ype] commands override this command.

l[ist] Prints (lists) all commands available. No explanation is given.

m[ail] *name* Mails a message to the specified users.

mb[ox] [*msglist*] Arranges for the given messages to end up in the standard mbox save file when mailx terminates normally. See mbox (ENVIRONMENT VARIABLES) for a description of this file. See also the ex[it] and q[uit] commands.

n[ext] [*message*] Goes to the next message matching *message*. A *msglist* may be specified, but in this case, the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pi[pe][*msglist*][*shell-command*]
| [*msglist*] [*shell-command*]
Pipes the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the cmd variable. If the page variable is set, a formfeed character is inserted after each message (see ENVIRONMENT VARIABLES).

pre[serve] [*msglist*]
ho[ld] [*msglist*] Preserves (Holds) the specified messages in the *mailbox*.

P[rint] [*msglist*]

T[ype] [*msglist*] Prints (Types) the specified messages on the screen, including all header fields. Overrides suppression of fields by the ig[nore] command.

p[rint] [*msglist*]
t[ype] [*msglist*] Prints (types) the specified messages. If crt is set, the messages longer than the number of lines specified by the crt variable are paged through the command specified by the PAGER variable. The default command is pg(1) (see ENVIRONMENT VARIABLES).

q[uit] Exits (Quits) from mailx, storing messages that were read in mbox and unread messages in the *mailbox*. Messages that have been saved explicitly in a file are deleted.

R[eply] [*msglist*]
R[espond] [*msglist*]
Responds to the author of each message in the *msglist*. The subject line is taken from the first message. If record is set to a filename, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

r[eply] [*message*]
r[espond] [*message*]
Replies to the specified message, including all other recipients of the message. If record is set to a filename, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

S[ave] [*msglist*] Saves the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the C[opy], fo[llowup], and F[ollowup] commands

and outfolder (ENVIRONMENT VARIABLES).

s[ave] [*filename*]
s[ave] [*msglist*] Appends the specified messages to the end of the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when mailx terminates unless keepsave is set (see also ENVIRONMENT VARIABLES and the ex[it] and q[uit] commands). mbox is the default filename.

se[t]
se[t] *name*
se[t] *name=string*
se[t] *name=number*
Defines (sets) a variable called *name*. The variable may be given a null, string, or numeric value. se[t] by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the mailx variables.

sh[ell] Invokes an interactive shell (see also SHELL (ENVIRONMENT VARIABLES)).

si[ze] [*msglist*] Prints the size in characters of the specified messages.

so[urce] *filename* Reads (sources) commands from the given file and return to command mode.

to[p] [*msglist*] Prints the top few lines of the specified messages. If the toplines variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

tou[ch] [*msglist*] Touches the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the mbox upon nor-

mal termination. See ex[it] and q[uit].

T[ype] [*msglist*]
P[rint] [*msglist*] Prints (types) the specified messages on the screen, including all header fields. Overrides suppression of fields by the ig[nore] command.

t[ype] [*msglist*]
p[rint] [*msglist*] Prints (types) the specified messages. If crt is set, the messages longer than the number of lines specified by the crt variable are paged through the command specified by the PAGER variable. The default command is pg(1) (see ENVIRONMENT VARIABLES).

u[ndelete] [*msglist*]
Restores (undeletes) the specified deleted messages. Will restore only those messages deleted in the current mail session. If autoprint is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

uns[et] *name* Causes the specified variables to be erased (unset). If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

ve[rsion] Prints the current version and release date.

v[isual] [*msglist*]
Edits the given messages with a (visual) screen editor. The messages are placed in a temporary file and the VISUAL variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

w[rite] [*msglist*] Writes the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the s[ave] com-

mand.

x[it]
ex[it] — Exits from mailx, without changing the *mailbox*. No messages are saved in the mbox (see also q[uit]).

z[+|-] — Scrolls the header display forward or backward one screenful. The number of headers displayed is set by the screen variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands may be entered only from input mode, by beginning a line with the escape character (tilde (~) by default). See escape under ENVIRONMENT VARIABLES for changing this special character.

~!*shell-command* — Escape to the shell.

~. — Simulate end-of-file (terminate message input).

~:*mail-command*

~_*mail-command* — Perform the command-level request. Valid only when sending a message while reading mail.

~? — Print a summary of tilde escapes.

~A — Insert the autograph string Sign into the message (see ENVIRONMENT VARIABLES).

~a — Insert the autograph string sign into the message (see ENVIRONMENT VARIABLES).

~b *name* . . . — Add the *names* to the blind carbon copy (bcc) list.

~c *name* . . . — Add the *names* to the carbon copy (cc) list.

~d — Read in the dead.letter file. See DEAD under ENVIRONMENT VARIABLES for a description of this file.

~e	Invoke the editor on the partial message. See also EDITOR under ENVIRONMENT VARIABLES.
~f [*msglist*]	Forward the specified messages. The messages are inserted into the message, without alteration.
~h	Prompt for Subject line and To, Cc, and bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.
~i *string*	Insert the value of the named variable into the text of the message. For example, ~A is equivalent to ~i Sign.
~m [*msglist*]	Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.
~p	Print the message being entered.
~q	Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in dead.letter. See DEAD (ENVIRONMENT VARIABLES) for a description of this file.
~r *filename*	
~< *filename*	
~< ! *shell-command*	Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.
~s *string* . . .	Set the subject line to *string*.
~t *name* . . .	Add the given *name* to the To list.
~v	Invoke a preferred (visual) screen editor on the partial message. See also VISUAL under ENVIRONMENT VARIABLES.

~w *filename*	Write the partial message onto the given file without the header.
~x	Exit as with ~q except the message is not saved in the DEAD file.
~| *shell-command*	Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and cannot be altered within mailx.

HOME= *directory*	The user's base of operations.
MAILRC= *filename*	The name of the startup file. Default is $HOME/.mailrc.

The following variables are internal mailx variables. They may be imported from the execution environment or set via the se[t] command at any time. The uns[et] command may be used to erase variables.

allnet	All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is noallnet. See also the alt[ernates] command and the metoo variable.
append	Upon termination, appends messages to the end of mbox file instead of at the top of mbox. Default is noappend (i.e., by default mailx saves messages at the top of mbox on exit).
askcc	Prompt for the Cc list after message is entered. Default is noaskcc.
asksub	Prompt for subject if it is not specified on the command line with the –s flag option. Enabled by default.
autoprint	Enable automatic printing of messages after d[elete] and u[ndelete] commands. Default is noautoprint.

`bang` Enable the special-casing of exclamation points (`!`) in shell escape command lines, as in `vi`(1). Default is `nobang`.

`cmd=`*shell-command*
Set the default command for the `pi[pe]` command. No default value.

`conv=`*conversion*
Converts `uucp` addresses to the specified address style. The only valid conversion now is `internet`, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also `sendmail` and the `-U` flag option.

`crt=`*number* Pipes messages having more than *number* lines through the command specified by the value of the `PAGER` variable (`pg`(1) by default). Disabled by default.

`DEAD=`*filename*
The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is `$HOME/dead.letter`.

`debug` Enables verbose diagnostics for debugging. Messages are not delivered. Default is `nodebug`.

`dot` Takes a period on a line by itself during input from a terminal as end-of-file. Default is `nodot`.

`EDITOR=`*shell-command*
The command to run when the `e[dit]` or `~e` command is used. Default is `ed`(1).

`escape=`*c* Substitute *c* for the `~` escape character.

`folder=`*directory*
The directory for saving standard mail files. User-specified filenames beginning with a plus (`+`) are expanded by preceding the filename with this directory name to obtain the real filename. If *directory* does not start with a slash (`/`), `$HOME` is prefixed to it. In order to use the plus (`+`) construct on a `mailx` command line, `folder` must be an exported `sh` environment variable. There is no default for the `folder` variable. See also `outfolder`,

later.

`header` Enables printing of the header summary when entering `mailx`. Enabled by default.

`hold` Preserves all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. Default is `nohold`.

`ignore` Ignores interrupts while entering messages. Handy for noisy dial-up lines. Default is `noignore`.

`ignoreeof` Ignores end-of-file during message input. Input must be terminated by a period (`.`) on a line by itself or by the `~.` command. Default is `noignoreeof`. See also `dot`, above.

`keep` When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

`keepsave` Keeps messages that have been saved in other files in the *mailbox* instead of deleting them. Default is `nokeepsave`.

`MBOX=`*filename*
The name of the file in which to save messages that have been read. The `x[it]` command overrides this function, as does saving the message explicitly in another file. Default is `$HOME/mbox`.

`metoo` If your login appears as a recipient, do not delete it from the list. Default is `nometoo`.

`LISTER=`*shell-command*
The command (and flag options) to use when listing the contents of the `folder` directory. The default is `ls`(1).

`onehop` When responding to a message that was originally sent to several recipients, the other recipient addresses normally are forced to be relative to the originating author's machine for the response. This flag option disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

`outfolder` Cause the files used to record outgoing messages to be located in the directory specified by the `fold-`

er variable, unless the pathname is absolute. Default is nooutfolder. See folder above and the S[ave], C[opy], fo[llowup], and F[ollowup] commands.

page Used with the pi[pe] command to insert a formfeed after each message sent through the pipe. Default is nopage.

PAGER=*shell-command*
The command to use as a filter for paginating output. This can also be used to specify the flag options to be used. Default is pg(1).

prompt=*string*
Set the command mode prompt to *string*. Default is ? .

quiet Refrain from printing the opening message and version when entering mailx. Default is noquiet.

record=*filename*
Record all outgoing mail in *filename*. Disabled by default. See also outfolder, above.

save Enable saving of messages in dead.letter on interrupt or delivery error. See DEAD for a description of this file. Enabled by default.

screen=*number*
Sets the number of lines in a screenful of headers for the h[eaders] command.

sendmail=*shell-command*
Alternate command for delivering messages. Default is mail(1).

sendwait Wait for background mailer to finish before returning. Default is nosendwait.

SHELL=*shell-command*
The name of a preferred command interpreter. Default is sh(1).

showto When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

`sign=`*string* The variable inserted into the text of a message when the `~a` (autograph) command is given. No default (see also `~i` (TILDE ESCAPES)).

`Sign=`*string* The variable inserted into the text of a message when the `~A` command is given. No default (see also `~i` (TILDE ESCAPES)).

`toplines=`*number*
The number of lines of header to print with the `to` (top) command. Default is 5.

`VISUAL=`*shell-command*
The name of a preferred screen editor. Default is `vi`(1).

FILES

`/usr/bin/mailx`	
`/usr/lib/mailx`	
`$HOME/.mailrc`	personal startup file
`$HOME/mbox`	secondary storage file
`/usr/mail/*`	post office directory
`/usr/lib/mailx/mailx.help*`	help message files
`/usr/lib/mailx/mailx.rc`	global startup file
`/tmp/R[emqsx]*`	temporary files

SEE ALSO

`biff`(1), `csh`(1), `ksh`(1), `mail`(1), `pg`(1), `sh`(1), `ls`(1), *A/UX Essentials* .

BUGS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be unset (`uns[et]`).

The full internet addressing is not fully supported by `mailx`. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a . are treated as the end of the message by `mail`(1) (the standard mail delivery program).

NAME

make — maintain, update, and regenerate groups of files

SYNOPSIS

make [-b] [-B] [-d*digits*] [-e] [-f *description-file*] [-g] [-i] [-k] [-K] [-n] [-p] [-P] [-q] [-r] [-s] [-t] [-u] [*target...*]

DESCRIPTION

make is used to maintain, update, and regenerate groups of files. The make program was designed to manage the systematic regeneration of programs and is typically used, but is not limited to, that purpose.

The actions of make are governed by a set of built-in rules. The user can supplement or replace these rules by providing an appropriate *description-file*.

The following is a brief description of all flag options:

-b Use compatibility mode for old description files. This mode is on by default.

-B Turn off compatibility mode.

-d*digits*
Debug mode. If specified without *digits*, full debug mode is invoked. If specified with a single digit, the specified debug subset is invoked; -d*digits* invokes each specified subset. Currently, subsets 0 and 1 are implemented.

-e Cause environment variables to override macro definitions within the description file.

-f *description-file*
Use the description file specified by *description-file*. A description file of – (hyphen) denotes the standard input.

-g Turn on additional capabilities to automatically checkout SCCS files. See "SCCS File Handling" below.

-i Ignore any error code that might be returned by a shell command. This mode can also be entered if the target .IGNORE: (see "Built-in Targets" below) appears in the description file.

-k If a shell command returns a nonzero status, abandon work on the current target, but continue to process other targets that do not depend on the abandoned target.

-K Turn off the -k flag option. The -K flag option is on by default. The -K flag option is most often used in a description file that invokes make, that is a member of a multi-level make hierarchy, and that is invoked by a top-level make with the -k flag option.

-n Print the commands in the description file as they would be executed, but do not actually execute them. Even lines beginning with an @ (at sign) are printed (see "Targets and Dependency Statements" below). However, if a command line has the string $(MAKE) in it, the line is always executed (see discussion of the $MAKEFLAGS macro under "Environment Variables and Macros" below).

-p Print out the built-in rules of make, including a complete set of macro definitions.

-P Search for Pre and Post files (see "MakeFile" below) in the directory /usr/lib. For example, for a description file named x.mk, make will search for and read /usr/lib/x.mkPre and /usr/lib/x.mkPost.

-q Question. The make command returns a zero or a nonzero status code depending on whether or not the target file is up-to-date.

-r Do not use the built-in rules of make. To do useful work, this flag option must be accompanied by an appropriate description file.

-s Silent mode. Do not print command lines before executing them. This mode is also entered if the target .SILENT: appears at any place in the description file.

-t touch(1) the target files (causing them to be up-to-date) without executing any commands.

-u Look for makecomm and Makecomm files in the user's home directory, as specified by the $HOME environment variable, and in the current directory. The search order is $HOME/makecomm, $HOME/Makecomm, ./makecomm and ./Makecomm. At most, one file from each directory is read by make. These files are read before any description files and can be used to define macros and rules.

Suffix List and Built-in Rules

make uses a *suffix list* and a set of *built-in rules* to determine how to regenerate a file. The suffix list and the built-in rules are based on the file naming requirements of the various software generation tools in the A/UX® environment. For example, a file whose suffix is .s is typically an assembly language program that is processed by the as command. A file whose suffix is .c is typically a C program that is processed by the cc command.

For example, if make is requested to regenerate a file called x.o, make examines the name of each file in the current directory and looks for all files that have a base name of x and a suffix. In this case, make finds the file x.c and then extracts and saves the suffix, .c. make then prepends the suffix to each member of the default suffix list, one at a time, and attempts to match the resulting string against each built-in rule, from the beginning of the rules to the end. If no match is found, make prepends the suffix to the next element in the suffix list. If no match is ever found, make concludes that it does not know how to regenerate the requested file. When a match is found, make executes the commands that are associated with the matched rule. In this case, the string .c.o matches the .c.o built-in rule. For the .c.o rule, the associated command is cc -O -c *base-name*.c. make then executes the command, which in this case generates the requested file, x.o.

For this version of make, the suffix list is, reading across the columns,

```
.obj        .obj~       .for        .for~
.pas        .pas~       .f          .f~
.o          .c          .c~         .y
.y~         .l          .l~         .s
.s~         .sh         .sh~        .h
.h~         .i
```

For this version of make, the built-in rules are, reading across the columns,

```
.c          .c~         .sh         .sh~
.c.o        .c~.o       .c~.c       .s.o
.s~.o       .c.i        .c~.i       .c.s
.y.o        .y~.o       .l.o        .l~.o
.y.c        .y~.c       .l.c        .c.a
.c~.a       .s~.a       .h~.h       .f.o
```

```
.f~.o      .p.o       .p~.o      .for.obj
.for~.obj  .pas.obj   .pas~.obj
```

The suffix list and built-in rules demonstrate three important features of make. First, the order of the suffix list and the built-in rules is extremely important because the order of both governs which rule will be used to process a file.

Second, the built-in rules demonstrate the right-most suffix member of a rule can be empty. This is true for the first four built-in rules. For example, the .c rule allows make to regenerate the file x, which has no suffix, from the file x.c.

Third, both the suffix list and the built-in rules contain the tilde (~) character. To make, the tilde character indicates an SCCS file (see sccsfile(4)). Because make was designed to parse suffixes, and SCCS files are identified by their s. prefix, make internally converts references of the form s.*filename* to *filename*~. Thus, the rule .c~.o would transform an SCCS C source file into an object file (.o).

By definition, a rule starts with a period (.) and cannot contain a slash (/). The format of a rule is:

.*target*: [:] [*dependency-list*]
<tab>[shell-command]
.
.
.

where *target* and at least one colon are required. Items enclosed in square brackets ([]) are optional.

Built-in rules cannot rely on another built-in rule to resolve a dependency. Only explicit dependencies can be listed in the dependency list of a rule.

Description Files

The built-in rules are often supplemented or overriden by the contents of a user-written *description file*. If the -f flag option is not present, the search order for description files is as shown below:

```
./makefile
./Makefile
./MakeFile
./s.makefile
./SCCS/s.makefile
./s.Makefile
```

```
./SCCS/s.Makefile
./s.MakeFile
./SCCS/s.MakeFile
```

The new description file, `MakeFile` is described under "`MakeFile` Pre- and Post-Processing" below. If the description file is –, the standard input is taken. More than one `-f` *description-file* argument pair may appear on the command line.

Include Files

If the string `include` or `Include` appears as the first seven characters of a line in a description file and is followed by a blank or tab character, `make` assumes that the rest of the line is the name of a description file, which is read by the current invocation of `make`, after macro substitution. The difference between `include` and `Include` is that if `include` does not exist, `make` will terminate with an error message. If `Include` does not exist, `make` will continue processing and will not issue an error message.

Targets and Dependency Statements

While `make` can use its built-in rules to perform simple regeneration tasks, `make` requires direction from the user to accomplish more complicated tasks, such as regenerating a program that is comprised of multiple source files. That direction is provided by the user in a description file from which `make` reads and processes user-written *dependency statements* to update one or more targets. The target is usually, but does not have to be, a program. If the dependency statement is incomplete, `make` uses its built-in rules to supplement the dependency statement.

The format for a dependency statement is nearly identical to the format for a built-in rule; a distinction is maintained so that if the user wishes to completely replace an existing built-in rule, the user can do so by providing the new rule in the description file. The format of a dependency statement is shown below:

target [*target*] : [:] [*dependency-list*] [;]
<tab>*shell-command*
.
.
.

Each *target* is an alphanumeric string separated by a blank. A target name cannot begin with a tab or a period (`.`) and cannot contain a colon (`:`) or a semicolon (`;`). At least one colon is re-

quired; a second colon is optional. The colon must be preceded by at least one target. The *dependency-list* is a blank-separated list of items on which the target depends. The items can be file names or other targets. The *shell-commands* are the commands that will be executed to update the target. Each shell command must be preceded by a tab character.

If a shell command is so long that it must be spread over two or more lines, the newline, which is automatically placed at the end of the each line by the standard text editors, can be escaped by a backslash (\).

Command lines are executed one at a time, each by its own shell. To have a series of commands executed by the same shell, append a semicolon (`;`) to the end of each shell command and escape any newlines, as previously described. This treatment of shell commands is particularly important for commands that are executed directly by the shell and whose result is effective only for the lifetime of the shell, such as the `cd` command.

By default, `make` hands shell commands to the Bourne shell, but if the `$SHELL` variable is set and exported in the user's environment, `make` hands shell commands to the specified shell.

`make` interprets the # character as the beginning of a comment. When parsing lines in a description file, `make` determines that the first line that does not begin with a tab or a # begins a new dependency statement or macro definition (see below).

The following example illustrates many of the concepts described above. In this dependency statement, the target is a program called `pgm`. `pgm` depends on two files `a.o` and `b.o`, and they in turn depend on their corresponding source files `a.c` and `b.c` and a common file `incl.h`:

```
#
# Making pgm.
#
pgm: a.o b.o
     cc a.o b.o -o pgm
a.o: incl.h a.c
     cc -c a.c;\
     echo "Done compiling a.c"
b.o: incl.h b.c
     cc -c b.c;\
     echo "Done compiling b.c"
```

make updates a target only if its dependents are newer than the target. For this example, make will generate a new a.o if the modification time of either incl.h or a.c is newer than the modification time of a.o or if a.o does not exist. The same is true of b.o.

Each line in a description file is terminated by a newline character. In the previous example, a semicolon is appended to the end of the line that invokes the C compiler and the newline is escaped by the a backslash character to pass both the invocation of the C compiler and the echo command to the same instance of the shell.

Because the built-in rules can be used to determine that a.c and b.c can be used to generate a.o and b.o respectively, the dependency statement for making pgm can be written more briefly:

```
pgm:  a.o b.o
      cc a.o b.o -o pgm
a.o b.o:    incl.h
```

When parsing shell commands, make first prints the command and then passes everything except the initial tab character directly to the shell as is. The following example uses "<tab>" to represent the tab character.

```
<tab>echo a\
<tab>b
```

When processed by make the following output will be produced:

```
<tab>echo a\
<tab>b
ab
```

The first and second lines are printed by make before the initial tab character is stripped and the third line is printed by the echo command.

Printing of the command before execution can be turned off by preceding each command with the @ character. For example, if the description file contains

```
@echo a\
b
```

the output of make will be

```
ab
```

Single-Colon and Double-Colon Targets

As described in the format for a dependency statement above, a dependency statement must have at least one colon, but may have an optional second colon. For the following dependency statement

```
a:      b.o
a:      c.o
a:      d.o
```

`make` concludes that `a` depends on `b.o`, `c.o`, and `d.o`. That is, the preceding dependency statements are equivalent to

```
a:      b.o c.o d.o
```

`make` treats dependency statements that have two colons differently so that a description file that contains

```
a::     b.o
a::     c.o
a::     d.o
```

contains three distinct and separate dependency statements. Whether a single-colon or double-colon rule, the target is updated whenever one of the following conditions is true:

Single-Colon
: if any dependent is newer than the target, or
if the target does not exist

Double-Colon
: if any dependent is newer than the target, or
if the target does not exist, or
if the target does not have a dependency list

Double-colon dependency statements are useful for those situations where a single target name is desired, but depending on the context, different commands need to be executed to update the target. For example:

```
a::     a.sh
        cp a.sh a

a::     a.c
        cc -o a a.c
```

The first dependency statement is executed if the current directory contains the file `a.sh`, and the second dependency statement is

executed if the current directory contains the file `a.c`. If both `a.sh` and `a.c` exist in the current directory, one or both target statements could be executed depending on whether the target is older than the dependents. If the target is missing, the target is considered to be older than the dependents.

Double-colon dependency statements are also useful for situtations in which a target has the same name as a subdirectory in the current directory. For example, if the dependency statement is

```
mail:
        cd mail;\
        cc mail.o -o mail
```

and the current directory contains a directory called `mail`, the commands to update the target will never be invoked. This is because `make` assumes that if the target `mail` exists (even if it is a directory) and has no dependency list, the target is up-to-date. As a result, the target name in single-colon dependency statement should never be the name of a subdirectory in the current directory. Often, however, assigning the target the same name as the directory that contains the files on which the target depends makes the description file more meaningful to the user. `make` processes double-colon dependency statements differently so that,

```
mail::
        cd mail;\
        cc mail.o -o mail
```

works as desired. This is because if a double-colon dependency statement has no dependency list, `make` processes the commands that up-date the target even if the target already exists. The caveat is, however, that `make` always recompiles `mail.c` even when the executable `mail` is newer than `mail.c`. A better solution to this problem is described under "Attributes" below.

As mentioned earlier, the user can replace a built-in rule by providing a new rule of the same name in the description file. The user can also disable a built-in rule as shown by the following example:

```
.c.a:;
```

`make` interprets a semicolon (`;`) that is not preceded by a command as a null command, which has the effect of disabling the specified rule.

Just as the built-in rules can be replaced, so can the default suffix list. The following line in a description file clears the suffix list:

```
.SUFFIXES:
```

The following line appends additional suffixes to the end of the existing suffix list:

```
.SUFFIXES: .n .x
```

Multiple suffix lists accumulate until cleared, as shown above, or `make` terminates.

Built-in Targets

The targets described in this section are actually built-in rules that enabled by the user by including them in a description file. If present, they modify the default behavior of `make`. Because `make` reads the entire description file before beginning to process dependency statements, the following built-ins, which must appear at the beginning of a line, are processed first, whether they appear at the beginning, middle, or end of the description file.

`.DEFAULT:`
: If a file must be made but there are no explicit shell commands or relevant built-in rules, the shell commands listed under `.DEFAULT:` are used.

`.IGNORE:`
: If present, `.IGNORE` has the same effect as the `-i` flag option, which is to ignore nonzero return codes from commands.

`.PRECIOUS:`
: The default behavior of `make` is to remove a target and its dependents when a quit or interrupt signal is received while processing the commands that update the target. Because the actions of `make` depend in large part on the mere existence of a file, removal of potentially incomplete files helps ensure that the proper files are regenerated each time. Removal can be avoided by making specific files dependent on `.PRECIOUS:`. See "Error Handling" below for further details.

`.MAKESTOP [` *exit-code* `]:`
: If present, `.MAKESTOP:` causes `make` to exit. `.MAKESTOP:` is useful in a multi-level directory and description file hierarchy to quickly by-pass a `make` in a particular directory or directories. *exit-code* is optional and defaults to zero if not specified. If no exit code is specified or if the specified exit

code is zero, make exits silently. If a nonzero exit code is specified, make prints a warning message.

.SILENT:
If present, .SILENT: has the same effect as the -s command line flag option.

Environment Variables and Macros

The documentation for make uses the term *macro* to name the entities that the shell documentation calls *environment variables.* A *macro* is a variable whose value is set in a description file and can be overridden from the make command line. Although make and the shell use these entities in nearly identical ways, there are differences, which are described below.

The following sample shell script

```
NAME=Joe
echo NAME
echo $NAME
```

produces the following result

```
NAME
Joe
```

The difference between the first and second echo commands is that first simply requests that the string NAME be echoed, while the second, by the prepended dollar sign ($) requests that the contents of NAME be echoed. Such a request is called *expansion.*

Expansion is handled differently in make. The following example description file

```
NAME=joe

all:
        echo NAME
        echo $NAME
```

produces the following result

```
NAME
AME
```

This is because make requires that macro names that are longer than one character be enclosed by parentheses or braces for expansion to occur. In this case, make sees the $ and attempts to expand a variable named N. No such variable is set, so nothing is

echoed and the `echo` command completes by echoing `AME`. The following description file produces the desired result:

```
NAME=joe

all:
        echo NAME
        echo $(NAME)
```

The use of braces is equivalent to the use of parentheses, so that `${NAME}` is equivalent to `$(NAME)`.

Each time `make` evaluates a macro, `make` strips one dollar sign (`$`) from it. Therefore, an extra dollar sign should be prepended to any macro that is part of a shell command line. When `make` is invoked, `make` reads the user's environment and makes all the variables found there available for modification by the description file.

Environment variables are processed before any description file and after the built-in rules; macro definitions in a description file override environment variables of the same name. The `-e` flag option causes environment variables to override macro definitions of the same name in a description file.

The formal definition of a macro is shown below:

macro-name = *string2*

By convention, macro names are upper-case. *macro-name* is an alphanumeric string that cannot contain a colon (`:`) or a semicolon (`;`). The equal sign (=) can be surrounded by spaces or tabs. *string2* is defined as all characters up to a comment character or an unescaped newline.

`make` provides several built-in macros. These macros are described below:

`MAKEFLAGS`
: If not present in the environment, `make` creates the `MAKEFLAGS` macro and assigns to it the flag options with which `make` was invoked. `MAKEFLAGS` is processed by `make` as containing any legal input flag option (except `-f`, `-p`, `-P`, `-r`, and `-u`) Thus, `MAKEFLAGS` always contains the current input flag options. This proves very useful for "super-makes". In fact, as noted above, when the `-n` flag option is used, the command `$(MAKE)` is executed anyway; hence, one can perform a `make -n` recursively on a whole software system to see what would have been executed. This

is because the -n is put in MAKEFLAGS and passed to further invocations of $(MAKE). This is one way of debugging all of the description files for a software project without actually causing the execution of update commands.

MAKELEVEL
: If not present in the environment, make creates the MAKELEVEL macro, assigns an initial value of zero, and exports it. If already present in the environment, make increments the value of MAKELEVEL by one. In this way, each subordinate invocation of make can know its level in a multi-level make hierarchy. This macro is read-only and cannot be modified by the description file.

MAKEBDIR
: If not present in the environment, make creates the MAKEBDIR macro and assigns to it the absolute pathname of the current directory. If already present in the environment, the value of MAKEBDIR is not changed. MAKEBDIR provides a way for each subordindate invocation of make to obtain the pathname of the top-level make.

MAKEGOALS
: For every invocation of make, make creates the MAKEGOALS macro and assigns to it the targets that were specified on the command line. For the command line,

```
$ make clean all clobber
```

MAKEGOALS will be set to "clean all clobber". If the current invocation of make invokes make, the invocation can be made as shown in the following example:

```
MAKE=make
cd dir; $(MAKE) $(MAKEGOALS)
```

In this way, the same command line arguments can be passed to subordinate invocations of make.

VPATH
: This version of make supports special processing of macro VPATH, if set. VPATH is useful for processing files that are located in a directory other than the current directory. In the following example, main.c is located in the current directory. func1.c is located in ../common, and func2.c is located in ../incl. make will search the directories specified by the VPATH variable for any dependencies that

are not in the current directory.

```
VPATH=../common:../incl
main: main.o func1.o func2.o
      cc -o $@ $>
```

In this example, `$@` (described below) expands to the target name and `$>` (described below) expands to the list of dependencies on the current target. If `main.c`, `func1.c`, or `func2.c` are not present in the current directory, `make` will use its built-in rules to search for SCCS versions of the files in the current directory (see "SCCS File Handling" below). If SCCS versions of the files are not found, `make` will search the pathnames specified by `VPATH`.

The following built-in macros define values for common software generation programs or flag options to those programs. Description files can replace or supplement the values of these macros to change the way in which the built-in rules work:

`AR` This macro is defined as `ar`.

`AS` This macro is defined as `as`.

`ASFLAGS`
This macro is defined as null and is provided as an argument to the assembler.

`CC` This macro is defined as `cc`.

`CFLAGS`
This macro is defined as `-O` and is provided as an argument to the C compiler.

`CHMOD`
This macro is defined as `chmod`.

`CP` This macro is defined as `cp`.

`F77`
This macro is defined as `f77`.

`F77FLAGS`
This macro is defined as null and is provided as an argument to the FORTRAN compiler.

`FORTRAN`
This macro is defined as `fortran`.

`FORTRANFLAGS`
This macro is defined as null and is provided as an argument

to the FORTRAN compiler.

GET
This macro is defined as `get` and is used to get SCCS versions of files.

GFLAGS
This macro is defined as null and is provided as an argument to `get`.

LD This macro is defined as `ld`.

LDFLAGS
This macro is defined as null and is provided as an argument to `ld`.

LEX
This macro is defined as `lex`.

LFLAGS
This macro is defined as null and is provided as an argument to `lex`.

MAKE
This macro is defined as `make`.

MV This macro is defined as `mv`.

PASCAL
This macro is defined as `pascal`.

PASCALFLAGS
This macro is defined as null and is provided as an argument to `pc`.

PC This macro is defined as `pc`.

PCFLAGS
This macro is defined as null and is provided as an argument to `pc`.

RM This macro is defined as `rm`.

YACC
This macro is defined as `yacc`.

YFLAGS
This macro is defined as null and is provided as an argument to `yacc`.

The following six built-in macros have special expansion capabilities that are useful for writing shell commands.

`*` The `*` macro stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for built-in rules.

`@` The `@` macro stands for the full target name of the current target. It is evaluated only for explicitly-named dependencies.

`<` The < macro is evaluated only for built-in rules or the `.DEFAULT` rule. It is the module that is out-of-date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the `.c.o` rule, the < macro would evaluate to the `.c` file. An example for making optimized `.o` files from `.c` files is:

```
.c.o:
cc -c -O $*.c
```

or:

```
.c.o:
cc -c -O $<
```

`?` The `?` macro is evaluated when explicit rules from the description file are evaluated. It is the list of prerequisites that are out of date with respect to the target; essentially, those modules that must be rebuilt.

`%` The `%` macro is only evaluated when the target is an archive library member of the form `lib(file.o)`. In this case, `@` evaluates to `lib` and `%` evaluates to the library member, `file.o`.

`>` The > macro is expanded to list all the dependencies on the current rule.

These six macros can have alternative forms. When an upper case `D` or `F` is appended to any of the six macros, the meaning is changed to "directory part" for `D` and "file part" for `F`. Thus, `$(@D)` refers to the directory part of the string `@`. If there is no directory part, `./` is generated.

The following description file demonstrates the use of the `?` and > macros in their standard and alternate forms:

```
pgm:
        @echo "? = $?"
```

```
        @echo "?D = $(?D)"
        @echo "?F = $(?F)"
        @echo "> = $>"
        @echo ">D = $(>D)"
        @echo ">F = $(>F)"

pgm:    dir/a.o
pgm:    dir/b.o
pgm:    dir/c.o
```

When `a.o` is the only object that is newer than the `pgm`, the following is output from `make`:

```
? = dir/a.o
?D = dir
?F = a.o
> = dir/a.o dir/b.o dir/c.o
>D = dir dir dir
>F = a.o b.o c.o
```

Macro Substitution

The contents of a macro can be substituted as shown below:

$(*macro-name* [: *subst1* =[*subst2*]])

are replaced by *string2*, which is delimited by blanks, tabs, newline characters and the beginning of lines. A substitute sequence can only replace the trailing characters of *subst1*. For example,

```
SAMPLE=/a/b/file.test

all:
        @echo "1     $(SAMPLE:file=FILE)"
        @echo "2     $(SAMPLE:test=TEST)"
        @echo "3     $(SAMPLE:a/=A/)"
        @echo "4     $(SAMPLE:b/file.test=K)"
        @echo "5     $(SAMPLE:a=A)
```

has the following output

```
1       /a/b/file.test
2       /a/b/file.TEST
3       /a/b/file.test
4       /a/K
5       /a/b/file.test
```

In the above example, only the second and fourth examples are

successful. The other examples fail because they do not substitute the trailing characters of the expanded macro.

The following example demonstrates the usefulness of string substitution:

```
all:  /u/test/a.o
      cc -S $(?:a=.c)
      mv $(?:.o=.s) .tmp
      sed "s/text/data/" > $(?:.o=.s) < .tmp
      as -o $@ $(?:.o=.s)
      rm .tmp
```

The above example uses the `@` (expand to the full target name of the current target) and the `?` (expand to the list of out-of-date dependencies) macros to produce the assembly language file for each dependent of the `all` target, change each occurrence of `text` to `data` using `sed`, and assemble each resulting `.s` file.

Substitution works only on macros that are part of shell command lines. This version of `make` does not support substitutions of macros that are part of dependency lists.

Macro Testing

This version of `make` supports the testing of macros, where the format is:

$(*macro-name*:*test-operator*)

The macro name may be set or unset and with or without an assigned value. The *test-operator* can be one of the following:

`L` The macro is expanded to the length of it's contents. An empty or null value expands to zero. This test operator is useful for determining whether or not to examine the contents of a macro.

`V` If the macro is set and has a non-null value, the macro is expanded to null; otherwise, the macro is expanded to `#`. This test can be used to control the execution of command lines as shown below:

$(*macro-name*:`V`) *conditional-command*

If the macro is not set, the macro is expanded to `#`, which causes `make` to evaluate the line as a comment. As a result, *conditional-command* is not executed.

N If the macro is set and has a non-null value, the macro is expanded to #; otherwise, the macro is expanded to null. This is the opposite of the V test operator described above and is used in the same way as the V test operator.

For example, assume the desire to have a target called `clean` if the macro `$CLNFILES` is set. The dependency statement would remove the files expanded from this macro. Here is how the dependency statement would look:

```
$(CLNFILES:V)clean:
$(CLNFILES:V)     @echo "Removing: $(CLNFILES)";\
        rm -f $(CLNFILES)
```

If the `$CLNFILES` macro is set and contains a non-null value, the `$(CLNFILES:V)` macro will be nulled out when `make` reads the description file, and the line is be processed just as if the description file contained:

```
clean:
        @echo "Removing: $(CLNFILES)";\
        rm -f $(CLNFILES)
```

The `$(CLNFILES)` macro is expanded just before the command line is executed. Macros that have test operators are expanded during the parsing of the command line. This means that the order of macros that have test operators is significant, which is unlike the normal behavior of macros that do not have test operators. Normal macros are expanded after all description files have been read and command line execution has begun. Expansion of macros that have test operators can be delayed by prefixing more $ characters, just as can be done with normal macros.

In the example above, notice that `$(CLNFILES:V)` does not appear in front of each line. This is because a single command line was used, and that command line was spread over two lines, with the newline escaped by the backslash (\) character. If there had been multiple command lines, each command line would have to have been preceded by a `$(CLNFILES:V)` macro.

Global Macros

This version of `make` provides special handling of user-defined macros that begin with `G_`. Such macros are automatically exported to the environment, so they can be easily passed to subordinate invocations of `make`.

Default Macro Values

This version of make supports a default expansion value for non-existent or null-valued macros. The default value is specified as shown below:

```
$(XYZ::default)
```

In this case, if the macro XYZ is not defined or has a null value, it is expanded to default. The default value is restricted to a single word, so the following example will not have the intended result

```
$(XYZ::This is the default)
```

Instead, XYZ is expanded to This. But the following example will work:

```
DEFAULT = This is the default

all:
        @echo "$(NOTDEFINED::$(DEFAULT))"
```

Pattern Matching on Macros

This version of make supports limited shell-style pattern matching on macros. For example,

```
cc -o $@ $(>:=*.o)
```

Each word in the expanded > macro is tested (shell-style) against the asterisk (*) pattern and compiled on a match.

Attributes

This version of make understands *attributes*, which can be placed before or after the dependents in a dependency list as shown below:

target: [*attributes*] [*dependents*] [*attributes*]

Attributes can be any of:

.FAKE
: If the target exists and has no dependents, the normal behavior of make for single-colon dependency statements is to do nothing. The addition of the .FAKE attribute to the dependency statement requires make to treat the target is if it did not exist. This, in turn, forces make to execute the associated commands.

.CURTIME
: This attribute causes make to use the current time rather

than the most recent modification time of the target, even if the target does not exist. This attribute is used with the .FAKE attribute to prevent the associated dependency statement from being invoked unless the dependents have been up-dated with a newer time.

.MAIN
The normal behavior of make when invoked without a target name on the command line is to search the description file for the first target, process the target, and then terminate. The addition of .MAIN to a dependency statement causes make to treat the associated dependency statement as if it were the first dependency statement in the description file.

.PRE
This attribute informs make that the associated target is to be made before any others, including .MAIN. Hence, this attribute can be used to place initialization commands. Because the entire description file is read before the targets are processed, the placement of this attribute is position-independent within the description file.

.POST
This attribute informs make that the associated target is to be processed after all others. Hence, this attribute can be used to place cleanup commands.

.KEEPTIME
This attribute causes make to maintain the target's original modification time, even after the target has been regenerared.

.OLDTIME
This attribute causes make to ignore the target's modification time and apply a modification time of 0 for the purpose of determining if the target should be up-dated. After the target is regenerated, make sets the correct modification time.

.NOVPATH
This attribute causes make to ignore the $VPATH macro for the associated target.

If targets that have .MAIN, .PRE, and .POST attributes are dependents of other targets, the targets are made in the order dictated by the dependencies and not by the attributes.

Attributes on double-colon dependency statements apply to all of them as a unit.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus `lib(file.o)` and `$(LIB)(file.o)` both refer to an archive library that contains `file.o`. (This assumes the `LIB` macro has been previously defined.) The expression `$(LIB)(file1.o file2.o)` is not legal. The built-in rules pertaining to archive libraries have the form *.suffix*`.a` where *suffix* is the suffix from which the archive member is to be made. An unfortunate by-product of the current implementation requires *suffix* to be different from the suffix of the archive member. Thus, one cannot have `lib(file.o)` depend upon `file.o` explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:
        lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date
.c.a:
        $(CC) -c $(CFLAGS) $<
        ar rv $@ $*.o
        rm -f $*.o
```

In fact, the `.c.a` rule listed above is built into `make` and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:
        lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        ar rv lib $?
        rm $?
        @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The `$?` list is defined to be the set of object file names (inside `lib`) whose C source files are out-of-date. The substitution mode translates the `.o` to `.c`. (Unfortunately, one cannot as yet transform to `.c~`; however, this may become possible in the future.) Note also, the disabling of the `.c.a:` built-in rule, which

would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

MakeFile Pre- and Post-Processing

If, by its search rules, make finds a description file named MakeFile or SCCS/s.MakeFile, the files /usr/lib/MakeFilePre and /usr/lib/MakeFilePost, if present, are read. If, for example, the description file is named x.mk and the -P flag option is specified, make will read /usr/lib/x.mkPre and /usr/lib/x.mkPost, if present.

The pre- and post-processing files are description files that can be used to store global environmental settings and rules for events that the user would like to occur prior to and after processing of a description file.

If the macro G_SGS_ROOT is present in the environment, make considers this to be the root for finding the pre- and post-processing files. For example, if G_SGS_ROOT is set to newroot, make will look for the pre- and post-processing files in the directory /newroot/usr/lib.

SCCS File Handling

As described in "Description Files" above, make can check-out read-only copies of makefile, Makefile, or MakeFile from an SCCS version of the file in the current directory or an SCCS version of the file located in a subdirectory named SCCS. This ability is separate from the built-in rules that govern the check-out of SCCS versions of dependents and is not turned off by the -r flag option to make.

The built-in rules for dependents can only check-out a file for which there is an SCCS version in the current directory; the built-in rules cannot check-out an SCCS version of a file located in a subdirectory named SCCS.

If the -g option is used, however, and the file does not exist in the current directory, make will first use its built-in rules to check-out the file in the current directory. If this attempt fails because the SCCS version does not exist, the -g flag option causes make to again search the current directory and then a subdirectory named SCCS, if present, and check out the file if found. Note that the current directory is searched twice, once by the built-in rules and once because the -g flag option was specified. Searching the

current directory makes the -g flag option especially useful with the -r flag option. The -r flag option turns off the built-in rules, and, thus, when used with the -g flag option, the current directory is search just once.

If a VPATH variable is present and set, the built-in rules are used to search the specified directories for SCCS versions of the file. If the -g flag option is specified, any subdirectories named SCCS in the VPATH directories are also searched.

The built-in rules for checking out dependents from their SCCS versions in the current directory are not used to process include files (see "Include Files" above). If the include file exists only in its SCCS version in the current directory, make will not check it out. If make is invoked with the -g flag option, however, make will check out an include file if it is present in an SCCS version in the current directory or in a subdirectory named SCCS.

In no case does make check out a copy of a description file, an include file, or a dependent file if the file already exists.

If a file is checked out by the built-in rules, make does not remove the checked out copy. If a file is checked out by the action of the -g flag option, the checked-out copy is automatically removed when no longer needed.

Error Handling

Shell commands that return a nonzero status normally terminate make. This behavior can be modified in a number of ways:

-i If present on the command line, this flag option causes make to ignore any nonzero status that is returned by a shell command and to continue processing the current description file.

.IGNORE:
See "Built-in Targets".

-k If present on the command line, this flag option causes make, to abandon work on the current target if a shell command returns a nonzero status. make will continue work on other targets in the description file that do not depend on the target for which the error was received.

- If prepended to any shell command, make will not terminate on an error that might occur as a result of executing the command, but will continue processing the description file. The - can be combined with the previously described @ symbol, which suppresses the printing of the command before it is

passed to the shell. The combination can be either -@ or @-. Both error messages returned by the shell and make's standard error messages are still printed.

As mentioned earlier, make removes a target and its dependents when a quit or interrupt signal is received. The following conditions apply:

1) The dependency statement must be a single-colon dependency statement. If the dependency statement is a double-colon dependency statement, make does not remove files.
2) The target must have existed before processing of the dependency statement began.
3) The target is not a dependent of the built-in target .PRECIOUS:.
4) make was not invoked with the -n flag option.
5) make was not invoked with the -t flag option.

The standard termination message due to a nonzero status code is shown below. In this case, make terminates because the description file, x.mk, does not exist in the directory /dir.

```
$ make -f x.mk
Make: Cannot read description file /dir/x.mk
Make: Stopped in directory /dir.
```

FILES

/bin/make
/usr/lib/*description-file*Pre
/usr/lib/*description-file*Post
$HOME/[Mm]akecomm
[Mm]akecomm
s.[Mm]ake[Ff]ile
SCCS/s.[Mm]ake[Ff]ile

SEE ALSO

cc(1), cd(1), csh(1), ksh(1), lex(1), sh(1), touch(1), yacc(1), printf(3S), sccsfile(4).

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

BUGS

The syntax `(lib(file1.o file2.o file3.o)` is illegal.

You cannot build `lib(file.o)` from `file.o`.

The macro `$(a:.o=.c~)` does not work.

NAME

makedev — prepare troff description files

SYNOPSIS

makedev *files*

DESCRIPTION

makedev reads description files about a particular device and converts them into a form suitable for reading by troff(1). Input to makedev is in the format described in font(5).

FILES

/usr/bin/makedev

SEE ALSO

troff(1), font(5).

A Typesetter-Independent troff, Brian W. Kernighan (Bell Laboratories, 1982).

Adventures with Typesetter-Independent troff, Mark Kahrs and Lee Moore (University of Rochester Technical Report 159, 1985).

NAME

makekey — generate encryption key

SYNOPSIS

```
/usr/lib/makekey
```

DESCRIPTION

makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and upper and lowercase letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 output key bits in the result.

makekey is intended for programs that perform encryption (e.g., ed(1) and crypt(1)). (The encryption scheme provided by them is not secure.) Usually, makekey's input and output will be pipes.

EXAMPLES

```
/usr/lib/makekey
abcdefgh23
23xq5GyrhLTCA
```

The first line invokes makekey, the second line is the input to makekey, and the third is the new key generated by makekey.

FILES

```
/usr/lib/makekey
```

SEE ALSO

crypt(1), ed(1), ex(1), passwd(4).

NAME

man — display the named manual page entries

SYNOPSIS

man [–c] [–d] [–T*term*] [–w] [*section*] *name* [*section name...*]

DESCRIPTION

man locates and prints an entry in the *A/UX Command Reference*, the *A/UX System Administrator's Reference*, or the *A/UX Programmer's Reference*. The *name* of the entry is entered in lowercase. The *section* number may not have a letter suffix. If no *section* is specified, the whole manual is searched for *name* and all occurrences of it are printed.

Note: If you specify a section number, only one section can be searched at a time.

Flag options and their meanings are

–T*term* Prints the entry as appropriate for terminal type *term*. For a list of recognized values of *term*, type `help term2`. The default value of *term* is `450`.

–w Prints on the standard output only the *pathnames* of the entries, relative to `/usr/catman`, or to the current directory for –d flag option.

–d Searches the current directory, rather than `/usr/catman`; requires the full filename (for example, `cu.1c`, rather than just `cu`).

–c Causes man to invoke col(1). Note that col(1) is invoked automatically by man unless *term* is one of the standard terminal types: `300`, `300s`, `450`, `37`, `4000a`, `382`, `4014`, `tek`, `1620`, or `X`.

man examines the environment variables `$TERM` and `$PAGER` (see `environ`(5)) and attempts to select flag options that adapt the output to the terminal being used. The `$PAGER` variable defaults to `more` if not set otherwise. The user may select `pg` with the appropriate flag options. The –T*term* flag option overrides the value of `$TERM`; in particular, one should use `–Tlp` when sending the output of man to a line printer.

section may be changed before each *name*.

EXAMPLES

The command

```
man man
```

would reproduce this entry on the terminal, as well as any other entries named `man` that may exist in other sections of the manual.

The command

```
man sync
```

searches through all sections to find the entry for `sync`. Since there is a `sync`(1) and a `sync`(2), both entries are provided. If you are looking only for the `sync` system call (found in section 2), specify the section number as follows:

```
man 2 sync
```

FILES

`/usr/bin/man`	
`/usr/catman/?_man/man[1-8]/*`	preformatted manual entries

SEE ALSO

`term`(4).

CAVEAT

The `man` command prints manual entries that were formatted by `nroff` and are printed using the correct terminal filters as derived from the `-T`*term* and `$TERM` settings. Typesetting or other non-standard printing of manual entries is not supported.

NAME

merge — three-way file merge

SYNOPSIS

merge [-p] *file1 file2 file3*

DESCRIPTION

merge incorporates all changes that lead from *file2* to *file3* into *file1*. The result goes to the standard output if -p is present, and into *file1* otherwise. merge is useful for combining separate changes to an original. Suppose *file2* is the original, and both *file1* and *file3* are modifications of *file2*. Then merge combines both changes.

An overlap occurs if both *file1* and *file3* have changes in a common segment of lines. merge prints information on how many overlaps occurred and includes both alternatives in the result. The alternatives are delimited as follows:

<<<<<<< *file1*
lines in file1
=======
lines in file3
>>>>>>> *file3*

If there are overlaps, the user should edit the result and delete one of the alternatives.

DISCLAIMER

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.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN 47907.

SEE ALSO

co(1), diff(1), diff3(1), rcsmerge(1).

NAME

mesg — permit or deny messages

SYNOPSIS

mesg [*choice*]

DESCRIPTION

mesg permits or denies receipt of messages sent by another user via write(1). With no argument, mesg reports the current permission state. The *choice* arguments may be either y or n.

mesg with choice n forbids messages by revoking nonuser write permission on the user's terminal. mesg with argument y reinstates permission.

EXAMPLES

```
mesg y
```

changes the permission to "yes" and the system reports:

```
Is Yes; Was No
```

or whatever is the current and former state of your message permission.

FILES

/bin/mesg
/dev/tty*

SEE ALSO

talk(1N), write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdir — make a directory

SYNOPSIS

mkdir *dirname ...*

DESCRIPTION

mkdir creates specified directories in mode 777 (possibly altered by the user's umask (see sh(1), csh(1) and ksh(1)). Standard entries, ., for the directory itself, and .., for its parent, are made automatically. These and other directories beginning with . are not visible in listings unless you use the –a flag option to ls.

mkdir requires write permission in the parent directory.

EXAMPLES

```
mkdir letters
```

creates a directory letters as a subdirectory of the current directory at the time you employ the command.

FILES

/bin/mkdir

SEE ALSO

chmod(1), csh(1), ksh(1), rm(1), rmdir(1), sh(1).

DIAGNOSTICS

mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns nonzero.

NAME

mkshlib — create shared library

SYNOPSIS

mkshlib -s *specs* [-n] -t *target* [-h *host*]

DESCRIPTION

mkshlib builds and maintains shared libraries. A shared library is similar in function to a normal, non-shared library. The primary differences appear at run time. The code in shared library routines can be used by more than one program at the same time. The executable code for a shared library, which is in the Common Object File Format (COFF), is accessed from the applications that call upon it by means of a special addressing structure provided within them during link edit.

(In contrast to the shared library, each program that makes use of a nonshared library gets a private copy of any library routines required.)

The shared library consists of two files (two sublibraries) containing source archives and executable object files, referred to as the host shared library and the target shared library, respectively. The host and target sublibraries may be on different systems. A host shared library is an archive which provides information used during link-edit (see ld(1) and ar(4)). The name of the host shared library is included on the cc(1) command line in the same way as a non-shared library (see cc(1)). All operations that can be performed on a non-shared library can be performed on a host shared library.

The target shared library contains the executable code for all the routines in the library and must be fully resolved. This library is brought into memory, if not already present, during execution of a program that calls upon it. The library is attached to a user's process during execution. The text section of target objects is shared by all processes using that target library, but each process gets its own copy of data.

The user interface to mkshlib consists of command line arguments and a shared-library specification file. The specification file provides information necessary to build the host and target shared libraries.

To build both sublibraries, provide both names. To build only the target library, do not provide a host name. (However, a host library is required to access the target library via the link edit process. Presumably, you either have a usable host library or will build one separately.) The `-n` option may be used to have `mkshlib` build only a new host shared library and reuse an existing target shared library. The name of the target library must be supplied, even if only the host is to be built.

To build the host and target files, `mkshlib` invokes other tools such as the archiver, `ar`(1), the assembler, `as`(1), and the loader, `ld`(1).

FLAG OPTIONS

The following command-line arguments are recognized:

`-s` *specs*
: Provide the name of the shared-library specification file, *specs*, which contains the information necessary to build the shared library. Its contents include a list of the object files to be included in the shared library, the branch-table specifications for the target library, the pathname directing where to install the target library, and the start addresses of text and data sections for the target library. Initialization specifications for imported variables are given in this file, if necessary. Imported variables are addresses external to the target shared library, such as the addresses of routines that the library may call upon. Details on the shared-library specification file are given after the command line arguments.

`-t` *target*
: Specify the name, *target*, of the target shared library to be produced. The location where the target library is to be installed is given in the specification file (see the `#target` directive below).

`-h` *host*
: Specify the name of the host shared library, *host*. If not specified, then the host shared library is not produced.

`-n` Do not generate a new target shared library. This option is used to update the host shared library only. The `-t` flag option and the target library name must still be supplied, because a version of the target shared library is needed to build the host shared library.

Shared Library Specification File

The specification file contains all the information necessary to build both the host and target shared libraries. The file contains directive names and associated specification information. Directive names must be at the start of the line. Some directives have specification information on the same line, and some directives introduce multiple specifications on following lines. Lines following such a directive are interpreted as specification lines for that directive, until another directive or the end of the file is encountered.

The six possible directive names and their use are described below. Directives may be given in any order in the specification file, except for the `#init` directive.

`##` *comment-text*

Specifies that the remainder of the line is a comment. All *comment-text* on that line is ignored. Comment lines may occur anywhere. Comments are optional, but recommended.

`#address` *section-name address*

Specify the start *address* in the virtual address space at which to bind *section-name* of the target shared-library executable. Typically, address directives are provided for the `.text` and `.data` sections of the target library. Addresses must be on a 256 kilobyte (KB) boundary.

The `.bss` section is grouped with the `.data` section, and does not require a start address.

`#branch`

branch-table-specification
branch-table-specification
branch-table-specification
.
.
.

All lines following the `#branch` directive are interpreted as branch-table specifications, until another directive is encountered. Only one `#branch` directive can be in a specification file. The branch table built from these specifications consists of jump instructions to the specified functions.

Branch-table specification lines have the following format:

function-name position

Only functions should be given branch-table entries, and those functions must be external. The position value is the relative location of the function name in the branch table. Each *function-name* may appear only once. The value of *position* for each *function-name* given is the position (or position range) of the name in the branch table. The value of *position* is a single integer, or a range of integers of the form *position1-position2*. (The use of a position range is given later.) Position values start with 1, each position value may be used only once, and all position values from 1 to the highest value used must be accounted for.

When adding functions to an existing library, provide the new functions at higher positions than in the existing branch table. Changing positions in an existing branch table renders that shared library not usable by previously linked applications.

A position range may also be used to reserve empty slots in the branch table for later use. Only the highest value of the range is associated with the function name. The remaining positions in the range may be used later for other functions.

`#init` *object*
initialization
initialization
initialization
.
.
.

Specify *object* with the name of an object file that requires initialization code (because it uses an imported variable). Each object file that requires initialization must be specified. (If the shared library being built is completely self-contained (uses no imported variable), then no `#init` directive is used, because no initialization code is necessary.)

All `#init` directives must be placed after the `#objects` directive and its associated specifications in the specification file.

An `#init` directive is followed by one or more *initialization* specification lines pertaining to the object file, *object*, named in the directive. Each line following the directive is interpreted as a specification line until another directive is encoun-

tered. Specify each line of *initialization* by using the following format:

import importptr

The placeholder *import* refers to an imported variable, and *importptr* is a pointer defined within the object file named in the `#init` directive preceding the initialization line. For each initialization line so specified, initialization code is generated in the form:

```
importptr = &import;
```

in which the value of `importptr` is set to the absolute address of `import`.

`#objects`
file
file
file
.
.
.

Specify each entry of *file* with the names of the object files constituting the target shared library.

This directive can be specified only once per shared library specification file. The lines following the directive are interpreted as specifications of *file* until another directive is encountered.

`#target` *pathname*
Specify the absolute *pathname* for the location of the target shared library on the target system. This pathname is copied into `a.out` files, and tells the operating system where to find the target shared library when executing a file that uses it. The maximum length for *pathname* is 64 characters.

FILES

`/usr/bin/mkshlib`	
`/lib/*_s.a`	Host (archive) library
`/shlib/*_s`	Target (executable) library
`/tmp/`*unique-name*	Temporary directory (name is PID and time)

SEE ALSO

ar(1), as(1), cc(1), ld(1), a.out(4), ar(4).
"Shared Libraries," in *A/UX Programming Languages and Tools, Volume I.*

NAME

mkstr — create an error message file by massaging C source

SYNOPSIS

mkstr [–] *messagefile prefix file ...*

DESCRIPTION

mkstr is used to create files of error messages. Using it can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

mkstr will process each of the specified *files*, placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name.

To process the error messages in the source to the message file mkstr keys on the string error(" in the input stream. Each time it occurs, the C string starting at the " is placed in the message file followed by a newline character and a null character; the null character terminates the message so it can be easily used when retrieved, the newline character makes it possible to sensibly cat the error message file to see its contents. The massaged copy of the input file then contains a lseek pointer into the file which can be used to retrieve the message, i.e.:

```
char efilname[] = "/usr/lib/pi_strings";
int  efil = -1;
```

You have to write the error-handling function yourself. The following is an example:

```
error(a1, a2, a3, a4)
{
        char buf[256];
        if (efil < 0) {
                efil = open(efilname, 0);
                if (efil < 0) {
oops:
                        perror(efilname);
                        exit(1);
                }
        }
        if (lseek(efil, (long) a1, 0) < 0L
        || read(efil, buf, 256) <= 0)
```

```
                goto oops;
        printf(buf, a2, a3, a4);
}
```

The optional - causes the error messages to be placed at the end of the specified message file for recompiling part of a large `mkstr`ed program.

EXAMPLES

If the current directory has files `a.c` and `b.c`, then

```
mkstr exs x *.c
```

would create a new file `exs` which holds all the error messages extracted from the source files `a.c` and `b.c`, as well as two new source files `xa.c` and `xb.c` which no longer contain the extracted error messages.

FILES

`/bin/mkstr`

SEE ALSO

`cc`(1), `xstr`(1), `lseek`(2).

BUGS

All the arguments except the name of the file to be processed are unnecessary.

NAME

mm — format documents that contain nroff and mm formatting requests mm macros

SYNOPSIS

mm [-T*tty-type*] [-12] [-c] [-e] [-t] [-E] [*file*...]

DESCRIPTION

Use mm to format documents using `nroff` and the mm(5) text-formatting macro package. It has options to specify preprocessing by `tbl`(1) or `eqn`/`neqn` (see `eqn`(1) and `neqn`(1)) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for `nroff` and mm are generated, depending on the options that you select.

Options for mm are given below. Any other arguments or flags (for example, `-rC3`) are passed to `nroff` as appropriate. You may use such options in any order, but you must put them before the *file*s arguments. If you do not specify arguments, mm prints a list of its options.

-T*tty-type* Specifies the type of output terminal; Here is a list of recognized values for *tty-type*.

`450`	prepares output for a DASI 450 (default for mm; also equivalent to `-T1620`).
`450-12`	prepares output for a DASI 450 in 12-pitch mode.
`300`	prepares output for a DASI 300 terminal.
`300-12`	prepares output for a DASI 300 in 12-pitch mode.
`300s`	prepares output for a DASI 300S.
`300s-12`	prepares output for a DASI 300S in 12-pitch mode.
`4014`	prepares output for a TEKTRONIX 4014.
`37`	prepares output for a TELETYPE+ Model 37 (default for `nroff`).
`382`	prepares output for a DTC-382.

`4000a`	prepares output for a TRENDATA 4000A.
`X`	prepares output for an EBCDIC line printer.
`hp`	prepares output for a Hewlett-Packard HP262x or HP264x (implies `-c`); also equivalent to `-T2621`, `-T2640`, and `-T2645`. `43` prepares output for a TELETYPE Model 43 (implies `-c`).
`40/4`	prepares output for a TELETYPE Model 40/4 (implies `-c`). also equivalent to `-T40/4`.
`745`	prepares output for a Texas Instrument 700 series terminal (implies `-c`); also equivalent to `-T735`.
`2631`	prepares output for a HP2631 printer (implies `-c`).
`2631-e`	same as `-T2631`, but in expanded mode.
`2631-c`	same as `-T2631`, but in compressed mode.
`832`	prepares output for an Anderson Jacobson 832 printer (implies `-c`).
`8510`	prepares output for a C. Itoh printer (implies `-c`).
`tn300`	prepares output for a Terminet 300 printer (implies `-c`).
`lp`	prepares output for a device with no reverse or partial line motions or other special features (implies `-c`). If you do not use this option, `mm` uses the value of the shell variable `$TERM` from the environment (see `profile`(4) and `environ`(5)) as the value of *tty-type*, if `$TERM` is set; otherwise, `mm` uses 450 as the value of *tty-type*. If you specify several terminal types, the last one takes precedence.

`-12` Indicates that the document is to be produced in 12-pitch. May be used when `$TERM` is set to one of `300`, `300s`, `450`, and `1620`. (The pitch switch on the DASI 300 and 300s terminals must be manually set to `12` if this option is used.)

`-c` Causes mm to invoke `col`(1); note that `col`(1) is invoked automatically by mm unless *term* is one of `300`, `300s`, `450`, `37`, `4000a`, `382`, `4014`, `tek`, `1620`, or `X`.

`-e` Causes mm to invoke `neqn`; also causes `neqn` to read the `/usr/pub/eqnchar` file (see `eqnchar`(5)).

`-t` Causes mm to invoke `tbl`(1).

`-E` Invokes the `-e` option of `nroff`.

As an example (assuming that the shell variable `$TERM` is set in the environment to `450`), the two command lines below are equivalent:

```
mm -t -rC3 -12 file
tbl file | nroff -cm -T450-12 -h -rC3
```

mm reads the standard input when - is specified instead of any filenames. (Mentioning other files together with - leads to disaster.) This option allows mm to be used as a filter, for example:

```
cat file | mm -
```

HINTS

1. mm invokes `nroff` with the `-h` flag. With this flag, `nroff` assumes that the terminal has tabs set every 8 character positions.
2. Use the `-o`*list* option of `nroff` to specify ranges of pages to be output. Note, however, that mm, if invoked with one or more of the `-e`, `-t`, and - options, *together* with the `-o`*list* option of `nroff` may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.
3. If you use the `-s` option of `nroff` (to stop between pages of output), use linefeed (rather than return or newline) to restart the output. The `-s` option of `nroff` does not work with the `-c` option of mm, or if mm automatically invokes `col`(1) (see `-c` option earlier).

4. If you lie to mm about the kind of terminal its output will be printed on, you will get (often subtle) garbage; however, if you are redirecting output into a file, use the `-T37` option, and then use the appropriate terminal filter when you actually print that file.

FILES

`/bin/mm`
`/usr/pub/terminals`

SEE ALSO

`checkmm`(1), `col`(1), `env`(1), `eqn`(1), `greek`(1), `mmt`(1), `nroff`(1), `tbl`(1), `troff`(1), `profile`(4), mm(5), `term`(5).
"mm Reference" in *A/UX Text Processing Tools.*

DIAGNOSTICS

mm "`mm: no input file`" if none of the arguments is a readable file and mm is not used as a filter.

NAME

mmt — typeset documents

SYNOPSIS

mmt [-a] [-e] [-t] [-p] [-g] [-T*tty-type*] [-D*dest*] [-z] [*file...*]

DESCRIPTION

mmt is very similar to mm(1), except that it typesets its input via troff (1) as opposed to formatting it via nroff(1). mmt uses the mm macro package and has options to specify preprocessing by tbl(1), pic(1), eqn(1), or grap(1). The proper pipelines and the required arguments and flags for troff(1) and for the macro packages are generated, depending on the options selected.

FLAG OPTIONS

The flag options are given below. Any other arguments or flags are passed to troff(1). Such options can occur in any order, but they must appear before *file*. If no arguments are given, mmt prints a list of its options.

-a	Send the output to an ASCII terminal.
-e	Invoke eqn(1) and cause eqn to read the /usr/pub/eqnchar file (see eqnchar(5)).
-t	Invoke tbl(1).
-p	Invoke pic(1).
-g	Invoke grap(1), which in turn calls pic(1).
-T*tty-type*	Create output for troff device *tty-type* (see troff(1)). The output is sent through the appropriate postprocessor (see daps(1)).
-D*dest*	Direct to output via device *dest*. The currently supported value for *dest* is: 4014 (TEKTRONIX 4014 terminal via the tc(1) filter).
-z	Invoke no output filter to process or redirect the output of troff(1).

mmt reads the standard input when – is specified instead of any filenames.

HINT

Use the -o*list* option of troff(1) to specify ranges of pages to be output. Note, however, that these commands, if invoked with one or more of the -e, -t, -p, -g, and – options, *together* with the -o*list* option of troff(1), may cause a harmless "broken

pipe'' diagnostic if the last page of the document is not specified in *list*.

FILES

`/bin/mmt`

SEE ALSO

`daps`(1), `env`(1), `eqn`(1), `mm`(1), `nroff`(1), `pic`(1), `tbl`(1), `tc`(1), `troff`(1), `profile`(4), `environ`(5), `mm`(5), `mv`(5).

A/UX Text Processing Tools.

DIAGNOSTICS

The message "`mmt: no input file`" is reported if none of the arguments is a readable file and the command is not used as a filter.

NAME

more, page — show the contents of a file in display-size chunks

SYNOPSIS

more [-c] [-d] [-f] [-l] [-*n*] [-s] [-u] [+*linenumber*] [*name* ...]

more [-c] [-d] [-f] [-l] [-*n*] [-s] [-u] [+/*pattern*] [*name* ...]

page *more-arguments*

DESCRIPTION

more is a filter which allows examination of continuous text one screenful at a time on a CRT terminal. It normally pauses after each screenful, printing --More-- at the bottom of the screen.

page functions similarly, except that the screen is cleared before each screenful is displayed (but only if a full screenful is displayed), and that k-1 rather than k-2 lines are printed in each screenful, where k is the number of lines the terminal can display.

If the user then presses RETURN, one more line is displayed. If the RETURN is preceded by an integer, that number becomes the new window size. If the user hits a space, another screenful is displayed. If a space is preceded by an integer, that number of lines is displayed. If the user presses d or CONTROL-D, 11 more lines (usually half a screenful) are displayed (a "scroll"). If d or CONTROL-D is preceded by an integer, that number becomes the new scroll size.

more looks in the file /etc/termcap to determine terminal characteristics and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

more looks in the environment variable MORE to preset any flags desired. For example, if you prefer to view files using the -c mode of operation, the sh command sequence

```
MORE='-c'; export MORE
```

or the csh command

```
setenv MORE -c
```

would cause all invocations of more, including invocations by programs such as man and msgs, to use this mode. (Note, however, that the man command also looks at the PAGER environment variable; see man(1).) Normally, the user will place the command

sequence that sets up the MORE environment variable in the shell startup file `.login`, `.profile`, or `.cshrc`.

If `more` is reading from a file rather than a pipe, then a percentage is displayed along with the `--More--` prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

The following flag options are available.

-n An integer which is the size (in lines) of the window which `more` will use instead of the default.

`-c` Causes `more` to draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while `more` is writing. This flag option will be ignored if the terminal does not have the ability to clear to the end of a line.

`-d` Causes `more` to prompt the user with the message

```
Hit space to continue, Rubout to abort
```

at the end of each screenful.

`-f` Causes `more` to count logical lines, rather than screen lines; that is, long lines are not folded. This flag option is recommended if `nroff` output is being piped through `ul`, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen positions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus `more` may think that lines are longer than they actually are, and, therefore, fold lines erroneously.

`-l` Causes `more` not to treat CONTROL-L (form feed) as special. If this flag option is not given, `more` will pause after any line that contains a CONTROL-L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

`-s` Squeezes multiple blank lines from the output, producing only one blank line. Especially helpful when viewing `nroff` output, this flag option maximizes the useful information present on the screen.

`-u` Suppresses normal processing of underlining. `more` will handle underlining such as produced by `nroff` in a manner appropriate to the particular terminal; if the terminal can perform

underlining or has a stand-out mode, more will output appropriate escape sequences to enable underlining or use stand-out mode for underlined information in the source file.

+*linenumber*
Causes more to start up at *linenumber*.

+/*pattern*
Causes more to start up two lines before the line containing the regular expression *pattern*, if the input is from a file. If input is from a pipe, more starts on the line where the pattern was found.

Once inside more, other sequences may be typed when more pauses. The sequences and their effects are as follows (*i* is an optional integer argument, defaulting to 1) :

= Displays the current line number.

v Starts up the editor vi at the current line (does not work if the input to the program is from a pipe).

h Help command; gives a description of all the more commands.

i:n
Skips to the *i*th next file given in the command line. (Skips to last file if *i* doesn't make sense.)

i:p
Skips to the *i*th previous file given in the command line. If this command is given in the middle of printing out a file, then more goes back to the beginning of the file. If *i* doesn't make sense, more skips back to the first file. If more is not reading from a file, the bell rings and nothing more happens.

:f Displays the current filename and line number.

:q or :Q
Exits from more (same as q or Q).

. Repeats the previous command.

*i*z Same as typing a space except that *i*, if present, becomes the new window size.

*i*s Skips *i* lines and prints a screenful of lines.

*i*f Skips *i* screenfuls and prints a screenful of lines.

*i*n Searchs for the *i*th occurrence of the last regular expression entered.

q or Q
Exits from more. The interrupt character may also be used.

i/*expr*
Searches for the *i*th occurrence of the regular expression *expr*. Terminated either by pressing RETURN or the ESCAPE key. If the input is a file (rather than a pipe), and there are fewer than *i* occurrences of *expr*, then the position in the file remains unchanged and an error message is printed. If the input is a file (rather than a pipe), and there are at least *i* occurrences of *expr*, a screenful is displayed, starting two lines before the place where the expression was found. If the input is a pipe and there are fewer than *i* occurrences of *expr*, an error message is printed and more exits (because the entire input stream has been read). If the input is a pipe and there are at least *i* occurrences of *expr*, a screenful is displayed, starting on the line where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

Goes to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file. (Doesn't work if the input to the program is from a pipe.)

!*command*
Invokes a shell with *command*. Terminated either by pressing RETURN or the ESCAPE key.

Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the --More--(*xx*%) message.

:!*command*
Invokes a shell with *command*. (Same as !*command*).

CONTROL-L (^L)
The user may redraw the screen by pressing CONTROL-L (^L). (This doesn't work if the input to the program is from a pipe.)

Any time output is being sent to the terminal, the user may press the quit key (normally CONTROL-\). more will stop sending output, and will display the usual --More-- prompt. The user may then enter one of the commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to noecho mode by this program so that the output can be continuous. What is typed will not show on the terminal, except for the / and ! commands.

If the standard output is not a terminal, then more acts just like cat, except that a header is printed before each file (if there is more than one).

EXAMPLES

```
nroff -ms +2 doc.n | more
```

would show the nroff output on the terminal screen.

FILES

/bin/more
/bin/page
/etc/termcap
/usr/lib/more.help

SEE ALSO

cat(1), pg(1), termcap(4), terminfo(4).

NAME

mt — magnetic tape manipulating program

SYNOPSIS

mt [-f*device-file*] *command* [*count*]

DESCRIPTION

mt is used to give commands to a magnetic tape drive. If *device-file* is not specified using the -f flag option, the environment variable TAPE is used; if TAPE does not exist, mt uses the device /dev/rmt12. Note that *special-file* must reference a raw (not block) tape device. By default, mt performs *command* once. *command* may be performed more than once by supplying a *count*.

command may be any one of the following set of commands. Only as many characters as are required to uniquely identify a command within the set need be specified.

eof, weof
: Write *count* end-of-file marks at the current position on the tape.

fsf
: Forward space *count* files.

fsr
: Forward space *count* records.

bsf
: Backspace *count* files.

bsr
: Backspace *count* records.

rewind
: Rewind the tape (*count* is ignored).

offline, rewoffl
: Rewind the tape and place the tape unit off-line (*count* is ignored).

format
: Format a tape cartridge (*count* is ignored). This only applies to /dev/rmt/tc*x*[n] device files that represent the Apple SC 40 Tape Backup.

status
: Print status information about the tape unit.

mt returns a 0 exit status when *count* invocations of *command* are successful, 1 if *command* was unrecognized, and 2 if any invocation of *command* failed.

FILES

/dev/rmt/*

SEE ALSO

mtio(4), dd(1), ioctl(2), environ(7).

NAME

mv — move or rename files

SYNOPSIS

mv [-i] [-f] [-] *file1 file2*

mv [-i] [-f] [-] *file... directory*

DESCRIPTION

mv moves (changes the name of) *file1* to *file2*.

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, mv prints the mode (see chmod(2)) and reads the standard input to obtain a line; if the line begins with y, the move takes place; if not, mv exits.

In the second form, one or more *files* (plain files or directories) are moved to the *directory* with their original file-names.

mv refuses to move a file onto itself.

FLAG OPTIONS

The following flag options are interpreted by mv:

- -i stands for interactive mode. Whenever a move is to supercede an existing file, the user is prompted by the name of the file followed by a question mark. If he answers with a line starting with y, the move continues. Any other reply prevents the move from occurring.
- -f stands for force. This flag option overrides any mode restrictions or the -i flag option.
- - means interpret all the following arguments to mv as file names. This allows file names starting with minus.

FILES

/bin/mv

SEE ALSO

cp(1), ln(1).

BUGS

If *file1* and *file2* lie on different file systems, mv must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

NAME

mvt — typeset view graphs and slides

SYNOPSIS

mvt [-a] [-e] [-t] [-p] [-g] [-T*tty-type*] [-D*dest*] [-z] [*file...*]

DESCRIPTION

This command is very similar to mmt(1), except that it typesets its input with the mv macro package for view graphs and slides. mvt has options to specify preprocessing by tbl(1), pic(1), eqn(1), grap(1). The proper pipelines and the required arguments and flags for troff(1) and for the macro package are generated, depending on the options selected.

FLAG OPTIONS

The flag options specific to mvt are given below. Any other arguments or flags are passed to troff(1). Options can occur in any order, but they must appear before *file*. If no arguments are given, mvt prints a list of its options.

-a	Send the output to an ASCII terminal.
-e	Invoke eqn(1) and cause eqn to read the /usr/pub/eqnchar file (see eqnchar(5)).
-t	Invoke tbl(1).
-p	Invoke pic(1).
-g	Invoke grap(1), which in turn calls pic(1).
-T*tty-type*	Create output for troff device *tty-type* (see troff(1)). The output is sent through the appropriate postprocessor (see daps(1)).
-D*dest*	Direct to output via device *dest*. The currently supported values for dest are: 4014 (TEKTRONIX 4014 terminal via the tc(1) filter).
-z	Invoke no output filter to process or redirect the output of troff(1).

mvt reads the standard input when – is specified instead of any file names.

HINT

Use the -o*list* option of troff(1) to specify ranges of pages to be output. Note, however, that these commands, if invoked with one or more of the -e, -t, -p, -g, and – options, *together* with the -o*list* option of troff(1), may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

FILES

/bin/mvt

SEE ALSO

daps(1), env(1), eqn(1), mm(1), mmt(1), nroff(1), pic(1), tbl(1), tc(1), troff(1), profile(4), environ(5), mm(5), mv(5).

A/UX Text Processing Tools.

DIAGNOSTICS

The message "mvt: no input file" is reported if none of the arguments is a readable file and the command is not used as a filter.

NAME

ndx — create a subject-page index for a document

SYNOPSIS

ndx *subjfile formatter-command-line*

DESCRIPTION

ndx, given a list of subjects (*subjfile*), searches a specified document and writes a subject-page index to the standard output.

subjfile is the list of subjects to be included in the index. Each subject must begin on a new line and have the following format.

word1 [*word2* ...] [, *wordn* ...]

Some examples are:

```
printed circuit boards
arrays
arrays, dynamic storage
Smith, W. P.
printed circuit boards, channel-oriented
Aranoff
University of Illinois
PL/I
```

The subject must start in column 1.

The syntax for *formatter-command-line* is

formatter [*option(s)*] *file(s)*

This is the command that is used to create the final form of the document. The following are examples of valid formatter command lines:

```
mm -Tlp files
nroff -mm -Tlp -rW60 file
troff -rB2 -Taps -r01.5i files
```

For more information about the formatter command line, see mm(1), mmt(1), nroff(1), and troff(1).

The document must include formatting commands for mm, nroff, or troff. The formatter command line tells ndx whether troff, nroff, mm, or mmt would be used to produce the final version of the document.

troff or mmt — Specify troff as the formatting program.

`nroff` or `mm` Specify `nroff` as the formatting program.

The options are those that would be given to the `troff`, `nroff`, `mm`, or `mmt` command in printing the final form of the document and are necessary to determine the correct page numbers for subjects as they are located in the document. `ndx` does not actually cause the final version of the document to be printed. The author must create the document separately. The indexer, of course, should not be used until the document is complete and no further changes are expected.

EXAMPLES

The command

```
ndx subjfile "nroff -mm -rW70 files" > indexfile
```

would produce a subject-page index for the document *files* and take its subjects from the list, *subjfile*. The page numbers would correspond to the document produced by

```
nroff -mm -rW70 files
```

The command

```
ndx subjfile "mm -rW60 -rN2 -rO0 ch1 ch2 ch3" > indexfile
```

would produce a subject-page index for the documents `ch1`, `ch2`, and `ch3`. The page numbers would correspond to the documents produced by

```
mm -rW60 -rN2 -rO0 ch1 ch2 ch3
```

The command

```
ndx subjfile "troff -rB2 -rW5i -rO1.5i -mm files" > indexfile
```

would produce a subject-page index for the document *file*. The page numbers would correspond to the document produced by

```
troff -rB2 -rW5i -rO1.5i -mm files
```

FILES

`/usr/bin/ndx`

SEE ALSO

mm(1), mmt(1), `nroff`(1), `subj`(1), `troff`(1).

NAME

neqn — format mathematical text for nroff

SYNOPSIS

neqn [-d*xy*] [-p*n*] [-s*n*] [-f*n*] [-] [*file...*]

DESCRIPTION

neqn is a preprocessor for typesetting mathematical text on typewriter-like terminals. Normal usage is:

neqn [*option*] *file* | nroff [*option*] | [*printer*]

If you do not specify files (or if you specify - as the last argument), neqn reads the standard input.

Full details of use are given in eqn(1).

FILES

/bin/neqn

SEE ALSO

eqn(1), mm(1), nroff(1), tbl(1), eqnchar(5), mm(5).
"eqn Reference" in *A/UX Text Processing Tools*.

NAME

netstat — show network status

SYNOPSIS

netstat [-Aan] [-f *address-family*] [*system*] [*core*]

netstat [-himnrs] [-f *address-family*] [*system*] [*core*]

netstat [-n] [-I *interface*] *interval* [*system*] [*core*]

DESCRIPTION

The netstat command symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. Using the third form, with an *interval* specified, netstat will continuously display the information regarding packet traffic on the configured network interfaces.

FLAG OPTIONS

The flag options have the following meaning:

-A	With the default display, show the address of any protocol control blocks associated with sockets; used for debugging.
-a	With the default display, show the state of all sockets; normally sockets used by server processes are not shown.
-h	Show the state of the IMP host table.
-i	Show the state of interfaces which have been autoconfigured (interfaces statically configured into a system, but not located at boot time are not shown).
-I *interface*	Show information only about this interface; used with an *interval* as described below.
-m	Show statistics recorded by the memory management routines (the network manages a private pool of memory buffers).
-n	Show network addresses as numbers (normally netstat interprets addresses and attempts to display them symbolically). This option may be

used with any of the display formats.

`-s` Show per-protocol statistics.

`-r` Show the routing tables. When `-s` is also present, show routing statistics instead.

`-f` *address-family*
Limit statistics or address control block reports to those of the specified *address family*. The following address families are recognized: `inet`, for `AF_INET`, *ns*, for `AF_NS`, and `unix`, for `AF_UNIX`.

The arguments, *system* and *core* allow substitutes for the defaults "`/vmunix`" and "`/dev/kmem`".

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form "`host.port`" or "`network.port`" if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases `/etc/hosts` and `/etc/networks`, respectively. If a symbolic name for an address is unknown, or if the `-n` flag option is specified, the address is printed numerically, according to the address family. For more information regarding the Internet "dot format," refer to `inet`(3N). Unspecified, or "wildcard," addresses and ports appear as "`*`".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network addresses of the interface and the maximum transmission unit ("mtu") are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route ("U" if "up"), whether the route is to a gateway ("G"), and whether the route was created dynamically by a redirect ("D"). Direct routes are created for each interface attached to the local host; the gateway field for such entries shows the address of the outgoing interface. The refcnt field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while

sending to the same destination. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When `netstat` is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column for the primary interface (the first interface found during autoconfiguration) and a column summarizing information for all interfaces. The primary interface may be replaced with another interface with the `-I` option. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

FILES

`/usr/bin/netstat`

SEE ALSO

`trpt`(1M), `hosts`(4), `networks`(4N), `protocols`(4N), `services`(4N).

BUGS

The notion of errors is ill-defined. Collisions mean something else for the IMP.

NAME

newform — change the format of a text file

SYNOPSIS

newform [-a*n*] [-b*n*] [-c*char*] [-e*n*] [-f] [-i*tabspec*] [-l*n*] [-o*tabspec*] [-p*n*] [-s] [*file...*]

DESCRIPTION

newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for -s, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like -e15 -l60 will yield results different from -l60 -e15.

FLAG OPTIONS

The following flag options are interpreted by newform:

-i*tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *tabspec* recognizes all tab specification forms described in tabs(1). In addition, *tabspec* may be --, in which newform assumes that the tab specification is to be found in the first line read from the standard input (see fspec(4)). If no *tabspec* is given, *tabspec* defaults to -8. A *tabspec* of -0 expects no tabs; if any are found, they are treated as -1.

-o*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for -i*tabspec*. If no *tabspec* is given, *tabspec* defaults to -8. A *tabspec* of -0 means that no spaces will be converted to tabs on output.

-l*n* Set the effective line length to *n* characters. If *n* is not entered, -l defaults to 72. The default line length without the -l option is 80 characters. Note that tabs and backspaces are considered to be one character (use -i to expand tabs to spaces).

-b*n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line

length (see -l*n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when -b with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform -l1 -b7 filename
```

The -l1 must be used to set the effective line length shorter than any existing line in the file so that the -b option is activated.

-e*n* Same as -b*n* except that characters are truncated from the end of the line.

-c*k* Change the prefix/append character to *k*. Default character for *k* is a space.

-p*n* Prefix *n* characters (see -c*k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

-a*n* Same as -p*n* except characters are appended to the end of a line.

-f Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* -o option. If no -o option is specified, the line which is printed will contain the default specification of -8.

-s Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by an * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

```
newform -s -i -l -a -e filename
```

DIAGNOSTICS

All diagnostics are fatal.

usage: ...
: newform was called with a bad option.

not -s format
: There was no tab on one line.

can't open file
: Self explanatory.

internal line too long
: A line exceeds 512 characters after being expanded in the internal work buffer.

tabspec in error
: A tab specification is incorrectly formatted, or specified tab stops are not ascending.

tabspec indirection illegal
: A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

EXIT CODES

0 – normal execution
1 – for any error

FILES

/bin/newform

SEE ALSO

csplit(1), tabs(1), fspec(4).

BUGS

newform normally only keeps track of printable characters; however, for the -i and -o options, newform will keep track of backspaces in order to line up tabs in the appropriate logical columns.

newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of -i - or -o -).

If the -f option is used, and the last -o option specified was -o-, and was preceded by either a -o- or a -i-, the tab specification format line will be incorrect.

NAME

newgrp — login to a new group

SYNOPSIS

newgrp [-] [*group*]

DESCRIPTION

newgrp changes a user's group identification. The user remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by newgrp, regardless of whether it terminated successfully or terminated due to an error condition (that is, unknown group).

Exported variables retain their values after invoking newgrp; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (PS1) other than $ (default) and has not exported PS1. After an invocation of newgrp, successful or not, their PS1 will now be set to the default prompt string $. Note that the shell command export (see sh(1)) is the method used to export variables so that their assigned value is retained when invoking new shells.

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

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

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in /etc/group as being a member of that group.

EXAMPLES

```
newgrp grpnam
```

would set the user's group ID to that of the group named grpnam.

FILES

/bin/newgrp
/etc/group
/etc/passwd

SEE ALSO

login(1), sh(1), group(4), passwd(4), environ(5).

BUGS

There is no convenient way to enter a password into /etc/group. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

news — display local news items

SYNOPSIS

news [-a] [-n] [-s] [*items*]

DESCRIPTION

news is used to keep the user informed of current events. By convention, these events are described by files in the directory /usr/news.

When invoked without arguments, news displays the contents of all current files in /usr/news, most recent first, with each preceded by an appropriate header. news stores the "currency" time as the modification date of a file named .news_time in the user's home directory (the identity of this directory is determined by the environment variable $HOME); only files more recent than this currency time are considered "current."

The -a flag option causes news to display all items, regardless of currency. In this case, the stored time is not changed.

The -n option causes news to report the names of the current items without displaying their contents, and without changing the stored time.

The -s flag option causes news to report how many current items exist, without displaying their names or contents, and without changing the stored time. It is useful to include such an invocation of news in one's .profile file, or in the system's /etc/profile.

All other arguments are assumed to be specific news items that are to be displayed.

If the interrupt character (usually CONTROL-c) is pressed during the display of a news item, the display stops and the next item is started. Another interrupt within one second of the first causes the program to terminate.

EXAMPLES

```
news
```

will display all files in /usr/news that have not been read previ-

ously by the account owner.

FILES

```
/bin/news
/etc/profile
/usr/news/*
$HOME/.news_time
```

SEE ALSO

profile(4), environ(5).

NAME

nice — run a command at low priority

SYNOPSIS

nice [*–increment*] *command* [*arguments*]

DESCRIPTION

nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed. The nice command built into the C shell is different from /bin/nice, which can be used by any shell.

The superuser may run commands with priority higher than normal by using a negative increment, e.g., --10.

EXAMPLES

For the Bourne shell (sh) or Korn shell (ksh):

```
nice -10 date
```

would cause the program date to be processed at a priority lower than normal (0), i.e., at +10. In the C shell (csh), the same is achieved by typing in

```
nice +10 date
```

FILES

/bin/nice

SEE ALSO

csh(1), ksh(1), nohup(1), sh(1), nice(2).

DIAGNOSTICS

nice returns the exit status of the subject command.

BUGS

An *increment* larger than 19 is equivalent to 19.

NAME

nl — line numbering filter

SYNOPSIS

nl [-b*type*] [-d*delim*] [-f*type*] [-h*type*] [-i*incr*] [-l*num*] [-n*format*] [-p] [-s*sep*] [-v*start#*] [-w*width*] *file*

DESCRIPTION

nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (for example, no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, nl assumes the text being read is in a single logical page body.

FLAG OPTIONS

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The flag options are:

-b*type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: a, number all lines; t, number lines with printable text only; n, no line numbering; p*string*, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is t (text lines numbered).

-h*type*	Same as -b*type* except for header. Default *type* for logical page header is n (no lines numbered).
-f*type*	Same as -b*type* except for footer. Default for logical page footer is n (no lines numbered).
-p	Do not restart numbering at logical page delimiters.
-v*start#*	*Start#* is the initial value used to number logical page lines. Default is 1.
-i*incr*	*incr* is the increment value used to number logical page lines. Default is 1.
-s*sep*	*sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.
-w*width*	*width* is the number of characters to be used for the line number. Default *width* is 6.
-n*format*	*format* is the line numbering format. Recognized values are: ln, left justified, leading zeroes suppressed; rn, right justified, leading zeroes suppressed; rz, right justified, leading zeroes kept. Default *format* is rn (right justified).
-l*num*	*num* is the number of blank lines to be considered as one. For example, -l2 results in only the second adjacent blank being numbered (if the appropriate -ha, -ba, and/or -fa option is set). Default is 1.
-d*xx*	The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the

second character remains the default character (:). No space should appear between the -d and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLES

```
nl -v10 -i10 -d!+ file1
```

will number `file1` starting at line number 10 with an increment of ten. The logical page delimiters are !+.

FILES

`/bin/nl`

SEE ALSO

`awk`(1), `cat`(1), `pr`(1), `sed`(1).

NAME

nm — display the symbol table of a common object file

SYNOPSIS

nm [-d] [-e] [-f] [-h] [-n] [-o] [-T] [-u] [-v] [-V] [-x] *file* ...

DESCRIPTION

The nm command displays the symbol table of each common object file *filename*. *filename* may be a relocatable or absolute common object file, or it may be an archive of relocatable or absolute common object files. nm prints the following information for each symbol. Note that the object file must have been compiled with the -g flag option of the cc(1) command for there to be *type*, *size*, or *line* information.

name
The name of the symbol.

value
Its value expressed as an offset or an address depending on its storage class.

class
Its storage class.

tv If the symbol is accessed through a transfer vector, this field contains *tv*.

type Its type and derived type. If the symbol is an instance of a structure or a union, the structure or union tag is given following the type (e.g., *struct-tag*). If the symbol is an array, the array dimensions are given following the type (e.g., char[n] [m]).

size Its size in bytes, if available.

line The source line number at which it is defined, if available.

section
For storage classes static and external, the object file section containing the symbol (e.g., text, data, or bss).

FLAG OPTIONS

The output of nm may be controlled using the following flag options:

-d Print the value and size of a symbol in decimal (the default).

`-o` Print the value and size of a symbol in octal instead of decimal.

`-x` Print the value and size of a symbol in hexadecimal instead of decimal.

`-h` Do not display the output header data.

`-v` Sort external symbols by value before they are printed.

`-n` Sort external symbols by name before they are printed.

`-e` Print only static and external symbols.

`-f` Produce full output. Redundant symbols (`.test`, `.data`, `.bss`), normally suppressed, are printed.

`-u` Print undefined symbols only.

`-V` Print the version of the `nm` command executing on the standard error output.

`-T` Truncate long names. By default, `nm` prints the entire name of the symbols listed. Since object files can have symbol names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow, forcing every column after the name to be misaligned. The `-T` flag option causes `nm` to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

Flag options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both `nm` *name* `-e -v` and `nm -ve` *name* print the static and external symbols in *name*, with external symbols sorted by value.

FILES

`/bin/nm`

SEE ALSO

`as`(1), `cc`(1), `ld`(1), `a.out`(4), `ar`(4).

WARNINGS

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the `-v` and `-n` flag options should be used only in conjunction with the `-e` flag option.

DIAGNOSTICS

`nm: `*name*`: cannot open`
name cannot be read.

`nm: `*name*`: bad magic`
name is not an appropriate common object file.

`nm: `*name*`: no symbols`
The symbols have been stripped from *name*.

NAME

nohup — run a command immune to hangups

SYNOPSIS

nohup *command* [*arguments*]

DESCRIPTION

nohup executes *command* immune to terminate (EOT, CONTROL-D) signal from the controlling terminal.

Note: nohup only operates in csh(1) and sh(1)).

With nohup, the priority is automatically incremented by 5. nohup should be used with processes running in background (with &) in order to prevent them from responding to interrupts or stealing the input from the next person who logs in on the same terminal. In csh, processes run in the background are automatically immune to hangups.

If output is not redirected by the user, both the standard output and standard error are sent to a file named nohup.out. If nohup.out is not writable in the current directory, output is redirected to $HOME/nohup.out.

EXAMPLES

```
nohup  nroff -mm  docsfile | lp
```

runs the nroff command shown, immune to hangups, quits, and interrupts.

It is frequently desirable to apply nohup to pipelines or lists of commands. This can be done only by placing pipelines and commands lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the nohup applies to everything in file. If the shell procedure file is to be executed often, then the need to type sh can be eliminated by giving file execute permission. Add an ampersand and the contents of file are run in the background with interrupts also ignored (see sh(1) and ksh(1)):

```
nohup file &
```

An example of what the contents of file could be is:

```
tbl ofile | eqn | nroff > nfile
```

FILES

/bin/nohup
nohup.out standard output and standard error file.

SEE ALSO

chmod(1), csh(1), ksh(1), nice(1), sh(1), nice(2), signal(3).

WARNINGS

nohup *command1 ; command2*
nohup applies only to *command1*

nohup (*command1 ; command2*)
is syntactically incorrect.

Be careful of where standard error is directed. The following command may put error messages on disk, making it unreadable:

```
nohup cpio -o < list > /dev/dsk/c8d0s0 2>&1 &
```

while the next command:

```
nohup cpio -o < list > /dev/dsk/c8d0s0 2>errors &
```

puts the error messages into the file errors.

NAME

nroff — text formatting language

SYNOPSIS

nroff [-o*list*] [-n*N*] [-s[*N*]] [-r*aN*] [-i] [-q] [-z] [-m*name*] [-T*tty-type*] [*file*...]

DESCRIPTION

nroff formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers.

An argument consisting of a minus (-) is taken to be a file name corresponding to the standard input.

FLAG OPTIONS

The flag options, which may appear in any order, but must appear before the *files*, are:

-o*list* Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. 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 BUGS below.)

-n*N* Number first generated page *N*.

-s*N* Stop every *N* pages. nroff will halt *after* every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a linefeed or newline (newlines do not work in pipelines, e.g., with mm(1)). This option does not work if the output of nroff is piped through col(1). When nroff (otroff) halts between pages, an ASCII BEL is sent to the terminal.

-r*aN* Set register *a* (which must have a one-character name) to *N*.

-i Read standard input after *files* are exhausted.

-q Invoke the simultaneous input-output mode of the .rd request.

-z Print only messages generated by .tm (terminal message) requests.

-m*name* Prepend to the input *files* the macro file /usr/lib/tmac/tmac.*name*.

-T*tty-type*

Prepare output for specified terminal. Known *tty-types* 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)
382	DTC-382
4000a	Trendata 4000a terminal (4000a is a synonym)
450	DASI-450 (Diablo Hyterm) printer
450-12	DASI-450 (Diablo Hyterm) printer set to 12-pitch (12 characters per inch)
832	Anderson Jacobson 832 terminal
8510	C.ITOH printer
lp	generic name for printers that can underline and tab (All text using reverse line feeds, such as those having tables, that is sent to lp must be processed with col(1))
tn300	GE Terminet 300 terminal
X	Printers equipped with TX print train

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

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

-u*n* Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

FILES

/bin/nroff	
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/nterm/*	terminal driving tables for nroff

/usr/pub/terminals list of supported terminals

SEE ALSO

checknr(1), col(1), deroff(1), greek(1), neqn(1), tbl(1). mm(1), mm(5).
"nroff/troff Reference" in *A/UX Text Processing Tools.*

BUGS

nroff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that nroff generates may be off by one day from your idea of what the date is.

When nroff is used with the –o*list* option inside a pipeline (e.g., with one or more of neqn(1), and tbl(1)), it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

NAME

nslookup — query name servers interactively

SYNOPSIS

nslookup
nslookup - *server*
nslookup *host-to-find* [*server*]

DESCRIPTION

nslookup is a program which queries DARPA Internet domain name servers.

server is a either the host name or address for a name server.

nslookup has two modes: interactive and non-interactive. Interactive mode allows the user to query the name server for information about various hosts and domains or print a list of hosts in the domain. Non-interactive mode is used to print just the name and Internet address of a host or domain.

Interactive mode is entered in the following cases:

a) when no arguments are given (the default name server will be used), and

b) when the first argument is a hyphen (–) and the second argument is the host name of a name server.

Non-interactive mode is used when the name of the host to be looked up is given as the first argument. The optional second argument specifies a *server*.

INTERACTIVE COMMANDS

Commands may be interrupted at any time by typing a CONTROL-C. To exit, enter the end-of-file signal, CONTROL-D. The command line length must be less than 80 characters.

Note: an unrecognized command will be interpreted as a host name.

host [*server*]
Look up information for *host* using the current default server, or using *server* if it is specified.

server *domain*

`lserver` *domain*
Change the default server to *domain*. `lserver` uses the initial server to look up information about *domain* while `server` uses the current default server. If an authoritative answer can't be found, the names of servers that might have the answer are returned.

`root`
Changes the default server to the server for the root of the domain name space. Currently, the host `sri-nic.arpa` is used. (This command is a synonym for the `lserver sri-nic.arpa`.) The name of the root server can be changed with the `set root` command.

`finger` [*name*] [> *filename*]
`finger` [*name*] [>> *filename*]
Connects with the finger server on the current host. The current host is defined when a previous lookup for a host was successful and returned address information (see the `set querytype=A` command). *name* is optional. > and >> can be used to redirect output in the usual manner.

`ls` *domain* [> *filename*]
`ls` *domain* [>> *filename*]
`ls -a` *domain* [> *filename*]
`ls -a` *domain* [>> *filename*]
`ls -h` *domain* [> *filename*]
`ls -h` *domain* [>> *filename*]
List the information available for *domain*. The default output contains host names and their Internet addresses. The `-a` option lists aliases of hosts in the domain. The `-h` option lists CPU and operating system information for the domain. When output is directed to a file, hash marks are printed for every 50 records received from the server.

`view` *filename*
Sorts and lists the output of the `ls` command with `more`(1).

`help`
`?` Prints a brief summary of commands.

`set` *keyword*[=*value*]
This command is used to change state information that affects the lookups. Valid keywords are:

`all`
Prints the current values of the various options to `set`. Information about the current default server and host is also printed.

`[no]debug`
Turn debugging mode on. A lot more information is printed about the packet sent to the server and the resulting answer.
(Default = `nodebug`, abbreviation = `[no]deb`)

`[no]defname`
Append the default domain name to every lookup.
(Default = `nodefname`, abbreviation = `[no]def`)

`domain=`*name*
Change the default domain name to *name*. The default domain name is appended to all lookup requests if the `defname` option has been set.
(Default = value in `/etc/resolv.conf`, abbreviation = `do`)

`querytype=`*value*
Change the type of information returned from a query to one of:

`A`	the host's Internet address (the default).
`CNAME`	the canonical name for an alias.
`HINFO`	the host CPU and operating system type.
`MD`	the mail destination.
`MX`	the mail exchanger.
`MG`	the mail group member.
`MINFO`	the mailbox or mail list information.
`MR`	the mail rename domain name.

Other types specified in the RFC883 document are valid but aren't very useful.

(Abbreviation = q)

[no]recurse
Tell the name server to query other servers if it does not have the information.
(Default = recurse, abbreviation = [no]rec)

retry=*number*
Set the number of retries to *number*. When a reply to a request is not received within a certain amount of time (changed with set timeout), the request is resent. The retry value controls how many times a request is resent before giving up.
(Default = 2, abbreviation = ret)

root=*host*
Change the name of the root server to *host*. This affects the root command.
(Default = sri-nic.arpa, abbreviation = ro)

timeout=*number*
Change the time-out interval for waiting for a reply to *number* seconds.
(Default = 10 seconds, abbreviation = t)

[no]vc
Always use a virtual circuit when sending requests to the server.
(Default = novc, abbreviation = [no]v)

TUTORIAL

The domain name space is tree-structured and currently has five top-level domains:

- com (for commercial establishments)
- edu (for educational institutions)
- gov (for government agencies)
- org (for not for profit orginizations)
- mil (for MILNET hosts)

If you are looking for a specific host, you need to know something about the host's organization in order to determine the top-level domain it belongs to. For instance, if you want to find the Internet address of a machine at UCLA, do the following:

a) Connect with the root server using the `root` command. The root server of the name space has knowledge of the top-level domains.

b) Since UCLA is a university, its domain name is `ucla.edu`. Connect with a server for the `ucla.edu` domain with the command `server ucla.edu`. The response will print the names of hosts that act as servers for the domain `ucla.edu`. Note that the root server does not have information about `ucla.edu` but knows the names and addresses of hosts that do. All future queries will be sent to the UCLA name server.

c) To request information about a particular host in the domain, type the host name. To request a listing of hosts in the UCLA domain, use the `ls` command. The `ls` command requires a domain name (in this case, `ucla.edu`) as an argument.

Note that if you are connected with a name server that handles more than one domain, all lookups for host name must be fully specified with its domain. For instance, the domain `harvard.edu` is served by `seismo.css.gov`, which also services the `css.gov` and `cornell.edu` domains. A lookup request for the host `aiken` in the `harvard.edu` domain must be specified as `aiken.harvard.edu`. However, the `set domain=`*name* and `set defname` commands can be used to automatically append a domain name to each request.

After a successful lookup of a host, use the `finger` command to see who is on the system or to finger a specific person. To get other information about the host, use the `set querytype=`*value* command to change the type of information desired and request another lookup. (`finger` requires *value* to be `A`.)

DIAGNOSTICS

If the lookup request was not successful, an error message is printed. Possible errors are:

`Time-out`
: The server did not respond to a request after a certain amount of time (changed with `set timeout=`*value*) and a certain number of retries (changed with `set retry=`*value*).

`No information`
: Depending on the query type set with the `set querytype` command, no information about the host was available, though the host name is valid.

`Non-existent domain`
The host or domain name does not exist.

`Connection refused`
`Network is unreachable`
The connection to the name or finger server could not be made at the current time. This error commonly occurs with `finger` requests.

`Server failure`
The name server found an internal inconsistency in its database and could not return a valid answer.

`Refused`
The name server refused to service the request.

`Format error`
The name server found that the request packet was not in the proper format. This error should not occur. It would indicate a bug in the program.

FILES

`/etc/bind/tools/nslookup`	
`/etc/resolv.conf`	initial domain name and name server addresses

SEE ALSO

`named`(1M), `resolver`(4).
RFC-882, RFC-883 (DNN Network Information Center, SRI International)

NAME

od — convert binary data to a displayable form in octal, decimal, hexadecimal, or ASCII

SYNOPSIS

od [-b] [-c] [-d] [-o] [-s] [-x] [*file*] [[+]*offset* [.][b]]

DESCRIPTION

od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing or an illegal flag option is specified, -o is default. The meanings of the format flag options are:

-b Interpret bytes in octal.

-c Interpret bytes in ASCII. Certain nongraphic characters appear as C escapes: null=\0, backspace=\b, form-feed=\f, newline=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.

-d Interpret words in unsigned decimal.

-o Interpret words in octal.

-s Interpret words in signed decimal.

-x Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The *offset* argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If . is appended, the offset is interpreted in decimal. If b is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by +.

Dumping continues until end-of-file. If a file contains many lines of repeating characters, od represents the repeating lines with an asterisk.

EXAMPLES

```
od -d file +2
```

produces an octal dump of file divided up into 32-bit words expressed in decimal equivalents with the dump starting point offset by 2 octal bytes.

FILES

/bin/od

SEE ALSO

adb(1), dump(1), nm(1), strings(1).

NAME

otroff — text formatting and typesetting

SYNOPSIS

otroff [-c*name*] [-k*name*] [-t] [-f] [-w] [-b] [-p*N*] [*file*...]

DESCRIPTION

otroff is the old version of troff(1). It formats text contained in *files* for standard output. The output is formatted for a Wang C/A/T phototypesetter.

FLAG OPTIONS

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (-) is taken to be a filename corresponding to the standard input. The flag options, which may appear in any order so long as they appear before the files, are:

-c*name* Insert before the input *files* the compacted macro files:

/usr/lib/macros/cmp.[nt].[dt].*name*
/usr/lib/macros/ucmp.[nt].*name*

-k*name* Compact the macros used in this invocation of otroff, placing the output in files [dt].*name* in the current directory

-t Direct output to the standard output, instead of the phototypesetter (this is the default).

-f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

-w Wait until phototypesetter is available, if it is currently busy.

-b Report whether the phototypesetter is busy or available. No text processing is done.

-p*N* Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.

EXAMPLES

The command

```
otroff -mm file
```

formats the text contained in *file*, and invokes the macro package mm.

FILES

/bin/otroff	
/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/tmp/trtmp*	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/font/dev*/*	font width tables for troff

SEE ALSO

cw(1), eqn(1), mmt(1), nroff(1), pic(1), tbl(1), tc(1), troff(1), mm(5), ms(5), mv(5).
A/UX Text Processing Tools.

NAME

pack, pcat, unpack — compress and expand files

SYNOPSIS

pack [–] [–f] *name* ...

pcat *name* ...

unpack *name* ...

DESCRIPTION

pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name*.z with the same access modes, access and modified dates, and owner as those of *name*. The -f flag option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If pack is successful, *name* will be removed. Packed files can be restored to their original form using unpack or pcat.

pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each .z file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;
the file name has more than 12 characters;
the file has links;
the file is a directory;

the file cannot be opened;
no disk storage blocks will be saved by packing;
the file is of zero length;
a file called *name*.z already exists;
the .z file cannot be created;
an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

pcat does for packed files what cat(1) does for ordinary files, except that pcat cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.z use:

```
pcat name.z
```

or just:

```
pcat name
```

To make an unpacked copy, say *nnn*, of a packed file named *name*.z (without destroying *name*.z) use the command:

```
pcat name > nnn
```

pcat returns the number of files it was unable to unpack. Failure may occur if:

the file name (exclusive of the .z) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of pack.

unpack expands files created by pack. For each file *name* specified in the command, a search is made for a file called *name*.z (or just *name*, if *name* ends in .z). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in pcat, as well as for the following:

a file with the "unpacked" name already exists;
if the unpacked file cannot be created.

EXAMPLES

```
pack file1
```

will pack file `file1` into `file1.z` and removes `file1` if packing is successful.

FILES

```
/usr/bin/pack
/usr/bin/pcat
/usr/bin/unpack
```

SEE ALSO

cat(1), compact(1).

See more(1)

NAME

pagesize — display system page size

SYNOPSIS

```
pagesize
```

DESCRIPTION

`pagesize` prints the size of a page of memory in bytes. This program is useful in constructing portable shell scripts.

FILES

```
/bin/pagesize
```

SEE ALSO

uvar(2).

NAME

passwd — change login password

SYNOPSIS

passwd [*name*]

DESCRIPTION

This command changes (or installs) a password associated with the login name.

Ordinary users may change only the password which corresponds to their login *name*.

passwd prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered passwd checks to see if the old password has aged sufficiently. If aging is insufficient, the new password is rejected and passwd terminates; see passwd(4).

Assuming aging is sufficient, a check is made to ensure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical, the cycle of prompting for the new password is repeated for at most two more times.

Passwords must meet the following requirements:

Each password must have at least six characters. Only the first eight characters are significant.

Each password must contain at least two alphabetic characters (uppercase or lowercase) and at least one numeric or special character.

Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

New passwords must differ from the old by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

One whose effective user ID is zero is called a superuser; see id(1), and su(1). Superusers may change any password; hence, passwd does not prompt superusers for the old password. Superusers are not forced to comply with password aging and password construction requirements. A superuser can create a null

password by entering a carriage return in response to the prompt for a new password.

EXAMPLES

```
passwd
```

will give the response

```
Changing password for <username>
```

and will then prompt for your present password and for the new password (twice).

FILES

```
/bin/passwd
/etc/passwd
```

SEE ALSO

chsh(1), login(1), id(1), su(1), crypt(3C), passwd(4).

NAME

paste — merge lines of several files or subsequent lines of one file

SYNOPSIS

paste *file1 file2 ...*
paste –d*list file1 file2 ...*
paste –s [–d*list*] *file1 file2 ...*

DESCRIPTION

In the first two forms, paste concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of cat(1) which concatenates vertically, i.e., one file after the other. In the last form above, paste replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the tab character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if – is used in place of a file name.

FLAG OPTIONS

The meanings of the flag options are:

–d Without this flag option, the newline characters of each but the last file (or last line in case of the –s flag option) are replaced by a tab character. This flag option allows replacing the tab character by one or more alternate characters (see below).

list One or more characters immediately following –d replace the default tab as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no –s flag option), the lines from the last file are always terminated with a newline character, not from the *list*. The list may contain the special escape sequences: \n (newline) \t (tab), \\ (backslash), and \0 (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use *–d* \\\\"").

–s Merge subsequent lines rather than one from each input file. Use tab, for concatenation, unless a *list* is specified with –d flag option. Regardless of the *list*, the very last character of the file is forced to be a newline.

– May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

```
ls | paste -d"" -
```

list directory in one column.

```
ls | paste - - - -
```

list directory in four columns.

```
paste -s -d"\ t\ n" file
```

combine pairs of lines into lines.

FILES

`/usr/bin/paste`

SEE ALSO

`cut`(1), `grep`(1), `pr`(1).

DIAGNOSTICS

`line too long`
Output lines are restricted to 511 characters.

`too many files`
Except for `-s` flag option, no more than 12 input files may be specified.

NAME

pax — copy files to or from an archive in an IEEE format

SYNOPSIS

pax [-cimopuvy] [-f *archive*] [-s *replstr*] [-t *device*] [*pattern*]...
pax -r [-cimnopuvy] [-f *archive*] [-s *replstr*] [-t *device*] [*pattern*]...
pax -w [-adimuvy] [-b *blocking*] [-f *archive*] [-s *replstr*] [-t *device*] [-x *format*] [*pathname*]...
pax -rw [-ilmopuvy] [-s *replstr*] [*pathname*]... *directory*

DESCRIPTION

pax reads and writes archive files that conform to the *"Archive/Interchange File Format"* specified in *IEEE Standard 1003.1-1988*. pax can also read, but not write, a number of other file formats in addition to those specified in the description *"Archive/Interchange File Format"*. Support for these traditional file formats, such as V7 tar and System V cpio, is provided for backward compatibility and to maximize portability.

pax also supports traditional cpio and tar interfaces if invoked with the name cpio or tar, respectively. See cpio(1) or tar(1) for more details.

Combinations of the -r and -w command line arguments specify whether pax reads, writes, or lists the contents of the specified archive, or moves the specified files to another directory.

The command-line arguments are:

-w Write the files and directories specified by *pathname* arguments to the standard output together with the pathname and status information prescribed by the archive format used. The placeholder *pathname* refers to a file or a directory. If it is a directory, pax recursively traverses all the files and subdirectories of *pathname* as well. If *pathname* is not given, then the standard input is read to get a list of pathnames to copy, one pathname per line. In this case, only those pathnames appearing on the standard input are copied.

-r Cause pax to read an archive file from the standard input. Only files with names that match any of the *pattern* arguments are selected for extraction. The selected files are conditionally created and copied relative to the

current directory tree, subject to the options described below. By default, the owner and group of selected files is that of the invoking process, and the permissions and modification times are the same as those in the archive.

The supported archive formats are automatically detected on input. The default output format is `ustar`, but may be overridden by the `-x` flag option described below.

`-rw` Cause `pax` to read the files and directories referred to in *pathname* and copy them to the destination *directory*. The placeholder *pathname* refers to the files and (recursively) subdirectories of that directory. If *pathname* is not given, the standard input is read to get a list of pathnames to copy, one pathname per line. In this case, only those pathnames appearing on the standard input are copied. The directory *directory* must exist and have the proper permissions before the copy can occur.

If neither the `-r` or `-w` options are given, then `pax` lists the contents of the specified archive. In this mode, `pax` lists normal files one per line, lists non-symbolic link pathnames as

pathname == linkname

and lists symbolic link pathnames, if supported by the implementation, as

pathname -> linkname

where *pathname* is the name of the file being extracted and *linkname* is the name of a file that appeared earlier in the archive.

If the `-v` option is specified, then `pax` list normal pathnames in the same format used by the *ls* utility with the `-l` option. Non-symbolic links are shown as

<listing> == linkname

and symbolic links, if supported, are shown as

<listing> -> linkname

pax is capable of reading and writing archives that span multiple physical volumes. Upon detecting an end-of-medium on an archive that is not yet completed, pax prompts the user for the next volume of the archive and allows the user to specify the location of the next volume.

pax exits with one of two types of values. If all of the files in the archive were processed successfully, pax exits with a value of 0. If pax aborted due to errors encountered during operation, pax exits with a nonzero value.

FLAG OPTIONS

The following flag options are available:

-a	Append the files specified by *pathname* to the specified archive.
-b *blocking*	Block the output at *blocking* bytes per write to the archive file. A k suffix multiplies *blocking* by 1024, a b suffix multiplies *blocking* by 512, and an m suffix multiplies *blocking* by 1048576 (1 megabyte). For machines with 16-bit integers the maximum buffer size is one byte less than 32 KB. If not specified, *blocking* is automatically determined on input and is ignored for -rw.
-c	Exclude the files and directories that match *pattern*.
-d	Do no create intermediate directories not explicitly listed in the archive. This option is ignored unless the -r option is specified.
-f *archive*	Specify the *archive* as the pathname of the input or output archive and override the default of standard input for -r or standard output for -w.
-i	Interactively rename files. Substitutions specified by -s options (described below) are performed before requesting the new filename from the user. A file is skipped if an empty line is entered and pax exits with an exit status of 0 if the *end-of-file* signal is encountered.
-l	Link files rather than copy them, when possible.
-m	Do not retain file modification times.

`-n` When `-r` is specified, but `-w` is not, the *pattern* arguments are treated as ordinary filenames. Only the first occurrence of each of these files in the input archive is read. The `pax` utility exits with an exit status of 0 after all files in the list are read. If one or more files in the list is not found, `pax` writes a diagnostic to standard error for each of the files and exits with a non-zero exit status. The file names are compared before any of the `-i`, `-s`, or `-y` options are applied.

`-o` Restore file ownership as specified in the archive. The invoking process must have appropriate privileges to accomplish this.

`-p` Preserve the access time of the input files after they have been copied.

`-s` *replstr* Modify filenames according to the substitution expression using the syntax of `ed`(1) as shown:

```
-s /old/new/[gp]
```

Any nonnull character may be used as a delimiter. For the example shown, / is used as the delimiter. Multiple `-s` expressions may be specified; the expressions are applied in the order specified, terminating with the first successful substitution. The optional trailing `p` causes successful mappings to be listed on standard error. The optional trailing `g` causes the `old` expression to be replaced each time it occurs in the source string. Files that substitute to an empty string are ignored both on input and output.

`-t` *device* Name the input or output archive device by using *device* as an implementation-defined identifier. This overrides the default of standard input for `-r` and standard output for `-w`.

`-u` Copy each file only if it is newer than a preexisting file with the same name. This implies `-a`.

`-v` List filenames as they are encountered. This produces a verbose list of the table of contents

on the standard output when both -r and -w are omitted; otherwise, the filenames are printed to standard error as they are encountered in the archive.

-x *format* Specify the output archive *format*. The input format, which must be one of the following, is automatically determined when the -r option is used. The supported formats are:

cpio The extended CPIO interchange format specified in *"Extended CPIO Format"* in *IEEE Standard 1003.1-1988*.

ustar The extended TAR interchange format specified in *"Extended TAR Format"* in *IEEE Standard 1003.1-1988*. This is the default archive format.

-y Interactively prompt for the disposition of each file. Substitutions specified by -s options (described above) are performed before prompting the user for disposition. The *eof* signal or an input line starting with the character q after the prompt causes pax to exit. Otherwise, an input line starting with anything other than y causes the file to be ignored. This option cannot be used in conjunction with the -i option.

Only the last of multiple -f or -t options take effect.

When writing to an archive, the standard input is used as a list of pathnames if *pathname* is not specified. The format is one pathname per line. Otherwise, the standard input is the archive file, which is formatted according to one of the specifications in *"Archive/Interchange File format"* in *IEEE Standard 1003.1-1988* or to some other implementation-defined format.

The user ID and group ID of the process, together with the appropriate privileges, affect the ability of pax to restore ownership and permissions attributes of the archived files. (See *format-reading utility* in *"Archive/Interchange File Format"* in *IEEE Standard 1003.1-1988*.)

The options -a, -c, -d, -i, -l, -p, -t, -u, and -y are provided for functional compatibility with the historical `cpio` and `tar` utilities. The option defaults were chosen based on the most common usage of these options, therefore, some of the options have meanings different than those of the historical commands.

Arguments

Various arguments are available for reading and writing archives:

directory — When copying files with `pax`, *directory* names the destination directory. To invoke this mode, use both the `-r` and `-w` options. The directory must exist and be writable before the copy or an error results.

pathname — The placeholder *pathname* is the file to be used as the source of a copy (or the file to be copied into an archive) instead of the files named on the standard input. When a directory is named, `pax` recursively copies all the files and subdirectories of *pathname* as well.

pattern — When reading data from an archive, *pattern* selects particular files in the standard shell-pattern matching notation. The default if *pattern* is not specified is to select all files.

EXAMPLES

The following command

```
pax -w -f /dev/rmt0 .
```

copies the contents of the current directory to tape drive 0.

The commands

```
mkdir newdir
cd olddir
pax -rw . newdir
```

copies the contents of `olddir` to `newdir`.

The command

```
pax -r -s ',/usr/*,,' -f pax.out
```

reads the archive `pax.out` with all files in the archive underneath `/usr` extracted relative to the current directory. For this example the substition string delimiter is the comma.

FILES

`/dev/tty` Used to prompt the user for information when the `-i` or `-y` options are specified

SEE ALSO

`cpio`(1), `find`(1), `tar`(1), `cpio`(4), `tar`(4).

DIAGNOSTICS

`pax` terminates immediately, without processing any additional files on the command line or in the archive, in the event of errors.

BUGS

Special permissions may be required to copy or extract special files.

Device, user ID, and group ID numbers larger than 65535 cause additional header records to be output. These records are ignored by some historical versions of `cpio`(1) and `tar`(1).

The archive formats described in *"Archive/Interchange File Format"* have certain restrictions that have been carried over from historical usage. For example, there are restrictions on the length of pathnames stored in the archive.

When getting an `ls -l` style listing on `tar` format archives, link counts are listed as 0 because the `ustar` archive format does not keep link-count information.

ADDITIONAL COPYRIGHT INFORMATION

See pack(1)

See machid(1)

NAME

pg — show the contents of a file in display-size chunks

SYNOPSIS

pg [*-number*] [*+linenumber*] [*+/pattern*] [-c] [-e] [-f] [-n] [-p *string*] [-s] [*file...*]

DESCRIPTION

The pg command is a filter which allows the examination of *files* one screenful at a time on a soft-copy terminal. (The file name - or null arguments indicate that pg should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, pg scans the terminfo(4) database for the terminal type specified by the environment variable TERM. If TERM is not defined, the terminal type dumb is assumed.

FLAG OPTIONS

The command line flag options are:

-number
: An integer specifying the size (in lines) of the window that pg is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

-p *string*
: Causes pg to use *string* as the prompt. If the prompt string contains a %d, the first occurrence of %d in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is :.

-c
: Home the cursor and clear the screen before displaying each page. This flag option is ignored if clear_screen is not defined for this terminal type in the terminfo(4) data base.

-e
: Causes pg *not* to pause at the end of each file.

-f
: Normally, pg splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The -f flag option inhibits pg from splitting lines.

`-n` Normally, commands must be terminated by a newline. This flag option causes an automatic end of command as soon as a command letter is entered.

`-s` Causes `pg` to print all messages and prompts in standout mode (usually inverse video).

+linenumber
Start up at *linenumber*.

+/pattern/
Start up at the first line containing the regular expression pattern. The terminal / may be omitted from this command.

The responses that may be typed when `pg` pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

`+1` newline
(or blank) This causes one page to be displayed. The address is specified in pages.

`+1 l`
With a relative address this causes `pg` to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

`+1 d` or CONTROL-d
Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or CONTROL-l
Typing a single period or CONTROL-l causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in ed(1) are available. They must always be terminated by a newline, even if the *–n* flag option is specified.

i/*pattern*/
Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around. The final / may be omitted unless m, b, or t modifiers are appended.

i^*pattern*^

i?*pattern*?
Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The final ^ and ? may be omitted from these commands unless the m, b, or t modifiers are appended. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, pg will normally display the line found at the top of the screen. This can be modified by appending m or b to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix t can be used to restore the original situation.

The user of pg can modify the environment of perusal with the following commands:

*i*f skip *i* screenfuls and print a screenful of lines *i*n Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i*p Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i*w Display another window of text. If *i* is present, set the window size to *i*.

*i*z same as typing a space except that *i*, if present, becomes the new window size.

s *filename*
Save the input in the named file. Only the current file being perused is saved. The white space between the s and *filename* is optional. This command must always be terminated by a newline, even if the *–n* flag option is specified.

h Help by displaying an abbreviated summary of available commands.

q or Q Quit pg.

! *command*
command is passed to the shell, whose name is taken from the SHELL environment variable. If this is not available, the default shell is used. This command must always be terminated by a newline, even if the *–n* flag option is specified.

At any time when output is being sent to the terminal, the user can press the quit key (normally CONTROL-\) or the interrupt key. This causes pg to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then pg acts just like cat(1), except that a header is printed before each file (if there is more than one).

EXAMPLES

A sample usage of pg in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, pg responds to the interrupt character (CONTROL-C by default) by terminating execution. Between prompts, however, the interrupt signal interrupts pg's current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

FILES

```
/usr/bin/pg
/usr/lib/terminfo/*
/tmp/pg*
```

SEE ALSO

crypt(1), ed(1), grep(1), more(1), terminfo(4).

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using pg as a filter with another command that changes the terminal I/O flag options (e.g., crypt(1)), terminal settings may not be restored correctly.

NAME

pic — troff preprocessor for drawing pictures

SYNOPSIS

pic [-T*tty-type*] [-] [*file...*]

DESCRIPTION

pic is a troff(1) preprocessor for drawing simple figures on a typesetter. The basic objects are boxes, lines, arrows, circles, ellipses, arcs, and text.

The optional argument -T*tty-type* specifies device *tty-type*; currently supported are psc (POSTSCRIPT® device such as the Apple LaserWriter®), iw (the Apple ImageWriter® II printer), and aps (Autologic APS-5). The default is -Tpsc.

FILES

/usr/bin/pic

SEE ALSO

grap(1), troff(1), postscript(4).
"pic Reference" in *A/UX Text Processing Tools.*

NAME

pr — format text for a print device

SYNOPSIS

pr [+*k*] [–*k*] [–a] [–d] [–e*ck*] [–f] [–h *head*] [–i*ck*] [–l*k*] [–m] [–n*ck*] [–o*k*] [–p] [–r] [–s*c*] [–t] [–w*k*] [*file*...]

DESCRIPTION

pr formats the named files on the standard output. If *file* is –, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the –s flag option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until pr has completed formatting.

FLAG OPTIONS

The flag options below may appear singly or be combined in any order:

+*k* Begin formatting with page *k* (default is 1).

–*k* Produce *k*-column output (default is 1). The flag options –e and –i are assumed for multicolumn output. Also, the –*k* flag option *must* be used if the –w (column width) flag option is used.

–a Print multicolumn output across the page.

–m Merge and format all files simultaneously, one per column (overrides the –*k*, and –a flag options).

–d Double-space the output.

–e*ck* Expand *input* tabs to character positions *k*+1, 2* *k*+1, 3* *k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any nondigit character) is given, it is treated as the input tab character (default for *c* is the tab character).

–i*ck* In *output*, replace white space wherever possible by inserting tabs to character positions *k*+1, 2* *k*+1, 3* *k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth posi-

tion are assumed. If *c* (any nondigit character) is given, it is treated as the output tab character (default for *c* is the tab character).

`-n`*ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of `-m` output. If *c* (any nondigit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

`-w`*k* For multicolumn output, set the width of a line to *k* character positions instead of the default 72 characters. This flag option *must* be used with the *-k* (number of columns) flag option.

`-o`*k* Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

`-l`*k* Set the length of a page to *k* lines (default is 66).

`-h` Use the next argument as the header instead of the file name.

`-p` Pause before beginning each page if the output is directed to a terminal (`pr` will ring the bell at the terminal and wait for a carriage return).

`-f` Use form feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

`-r` Print no diagnostic reports on failure to open files.

`-t` Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit formatting after the last line of each file without spacing to the end of the page.

`-s`*c* Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

EXAMPLES

```
pr -3dh "file list" file1 file2
```

formats `file1` and `file2` as a double-spaced, three-column listing headed by "`file list`".

```
pr -e9 -t < file1 > file2
```

writes `file1` on `file2`, expanding tabs to columns 10, 19, 28, 37, ...

FILES

```
/bin/pr
/dev/tty*
```

SEE ALSO

`cat`(1), `fmt`(1), `lp`(1), `lpr`(1).

NAME

printenv — display the value of variables set in the current environment

SYNOPSIS

printenv [*argument*]

DESCRIPTION

printenv takes an environment variable name as an *argument* and displays only the value of that variable. If no *argument* is given, it displays the values for the entire environment.

Examples of environment variable names are:

HOME	pathname of user's home directory.
SHELL	the shell present at login.
PATH	search path for binary programs.
TERM	type of terminal used.
LOGNAME	the login name of the user.
TERMCAP	terminal capabilities string.
EXINIT	a startup list of commands read by ex, edit and vi.

The man page on the shell you are using (csh (1), ksh (1), or sh (1)) gives a complete list of the environment variables that apply to you.

EXAMPLES

```
printenv HOME
```

displays the pathname of your home directory.

FILES

/bin/printenv

SEE ALSO

csh(1), env(1), ksh(1), sh(1), stty(1), tset(1), environ(5).

NAME

prof — display profile data

SYNOPSIS

prof [-a] [-c] [-g] [-h] [-m *mdata*] [-n] [-o] [-s] [-t] [-x] [-z] [*prog*]

DESCRIPTION

prof interprets the profile file produced by the monitor(3C) function. The symbol table in the object file *prog* (a.out by default) is read and correlated with the profile file (mon.out by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

FLAG OPTIONS

The mutually exclusive flag options t, c, a, and n determine the type of sorting of the output lines:

-t Sort by decreasing percentage of total time (default).

-c Sort by decreasing number of calls.

-a Sort by increasing symbol address.

-n Sort lexically by symbol name.

The mutually exclusive flag options o and x specify the printing of the address of each symbol monitored:

-o Print each symbol address (in octal) along with the symbol name.

-x Print each symbol address (in hexadecimal) along with the symbol name.

The following flag options may be used in any combination:

-g Include nonglobal symbols (static functions).

-z Include all symbols in the profile range (see monitor(3C)), even if associated with zero number of calls and zero time.

-h Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)

-s Print a summary of several of the monitoring parameters and statistics on the standard error output.

-m *mdata*
Use file *mdata* instead of mon.out for profiling data.

For the number of calls to a function to be tallied, the –p flag option of cc(1) must have been given when the file containing the function was compiled. This flag option to the cc command also arranges for the object file to include a special profiling start-up function that calls monitor(3C) at the beginning and end of execution. It is the call to *monitor* at the end of execution that causes the mon.out file to be written. Thus, only programs that call exit(2) or return from *main* cause the mon.out file to be produced.

FILES

/bin/prof	
mon.out	for profile
a.out	for namelist

SEE ALSO

cc(1), nm(1), exit(2), profil(2), monitor(3C).

BUGS

There is a limit of 600 functions that may have call counters established during program execution. If this limit is exceeded, other data is overwritten and the mon.out file is corrupted. The number of call counters used is reported automatically by the prof command whenever the number exceeds 250.

NAME

prs — display information about an SCCS file

SYNOPSIS

prs [-a] [-c [*date-time*]] [-d [*dataspec*]] [-e] [-l] [-r [*SID*]] *file* ...

DESCRIPTION

prs displays, on the standard output, parts or all of an SCCS file (see sccsfile(4)) in a user-supplied format. If a directory is named, prs behaves as though each file in the directory were specified as a named file, except that nonSCCS 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 or directory to be processed; nonSCCS files and unreadable files are silently ignored.

Arguments to prs, which may appear in any order, consist of *keyletter* arguments and filenames.

All the described *keyletter* arguments apply independently to each named file:

-d[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see DATA KEYWORDS) interspersed with optional user-supplied text.

-r[*SID*] Used to specify the SCCS Identification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed. The format for the date is: *mm/dd/yy* [*hh:mm:ss*].

-e Requests information for all deltas created *earlier* than and including the delta designated via the -r keyletter or the date given by the -c flag option.

-l Requests information for all deltas created *later* than and including the delta designated via the -r keyletter or the date given by the -c flag option.

-c[*date-time*] Cutoff date-time, in the form: *YY*[*MM*[*DD*[*HH*[*MM*[*SS*]]]]]. Units omitted from the the date-time default to their maximum possible values; that is, -c7502 is equivalent to -c750228235959. Any number of nonnumeric

characters may separate the various 2-digit pieces of the *cutoff* date in the form: `"-c77/2/2 9:22:25"`.

`-a` Requests information for both removed, i.e., delta type = *R*, (see `rmdel(1)`) and existing, i.e., delta type = *D*, deltas. If the `-a` keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see `sccsfile(4)`) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information displayed by `prs` consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a RETURN.

"User-supplied text" is any text other than recognized data keywords. A tab is specified by `\t` and RETURN/newline is specified by `\n`. The default data keywords are:

```
:Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:
```

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
`:Dt:`	Delta information	Delta Table	See below*	S
`:DL:`	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
`:Li:`	Lines inserted by Delta	"	nnnnn	S
`:Ld:`	Lines deleted by Delta	"	nnnnn	S
`:Lu:`	Lines unchanged by Delta	"	nnnnn	S
`:DT:`	Delta type	"	D or R	S
`:I:`	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
`:R:`	Release number	"	nnnn	S
`:L:`	Level number	"	nnnn	S
`:B:`	Branch number	"	nnnn	S
`:S:`	Sequence number	"	nnnn	S
`:D:`	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
`:Dy:`	Year Delta created	"	nn	S
`:Dm:`	Month Delta created	"	nn	S

:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes or no*	*S*
:MP:	MR validation pgm name	"	*text*	*S*
:KF:	Keyword error/warning flag	"	*yes or no*	*S*
:KV:	Keyword validation string	"	*text*	*S*
:BF:	*Branch flag*	"	*yes or no*	*S*
:J:	*Joint edit flag*	"	*yes or no*	*S*
:LK:	*Locked releases*	"	*:R:...*	*S*
:Q:	*User defined keyword*	"	*text*	*S*
:M:	*Module name*	"	*text*	*S*
:FB:	*Floor boundary*	"	*:R:*	*S*
:CB:	*Ceiling boundary*	"	*:R:*	*S*
:Ds:	*Default SID*	"	*:I:*	*S*
:ND:	*Null delta flag*	"	*yes or no*	*S*
:FD:	File descriptive text	*Comments*	*text*	*M*
:BD:	*Body*	*Body*	*text*	*M*
:GB:	*Gotten body*	"	*text*	*M*
:W:	A form of `what`(1) string	*N/A*	*:Z::M:\t:I:*	*S*
:A:	A form of `what`(1) string	*N/A*	*:Z::Y: :M: :I::Z:*	*S*
:Z:	`what`(1) string delimiter	*N/A*	*@(#)*	*S*
:F:	*SCCS file name*	*N/A*	*text*	*S*
:PN:	*SCCS file path name*	*N/A*	*text*	*S*

* *:Dt: = :DT: :I: :D: :T: :P: :DS: :DP:*

EXAMPLES

The command

```
prs -d"User IDs for :F: are:\n:UN:" s.file
```

may produce on the standard output:

```
User IDs for s.file are:
```

```
xyz
131
abc
```

The command

```
prs -d"Newest delta for pgm :PM:: \
        :I: Created :D: By :P:" -r s.file
```

may produce on the standard output:

```
Newest delta for pgm main.c: C.7 Created 77/12/1 By cas
```

As a *special case*,

```
prs s.file
```

may produce on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the –a keyletter.

FILES

/usr/bin/prs
/tmp/pr?????

SEE ALSO

admin(1), cdc(1), comb(1), delta(1), get(1), help(1), rmdel(1), sact(1), sccs(1), sccsdiff(1), unget(1), val(1), what(1), sccsfile(4).
"SCCS Reference" in *A/UX Programming Languages and Tools, Volume 2*.

DIAGNOSTICS

Use help(1) for explanations.

NAME

ps — report process status

SYNOPSIS

ps [-e] [-d] [-a] [-f] [-l] [-c*corefile*] [-s*swapdev*]
[-n*namelist*] [-t*termlist*] [-p*proclist*] [-u*uidlist*] [-g*grplist*]

DESCRIPTION

ps prints certain information about active processes. Without flag options, information is printed about processes associated with the current terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of flag options.

Flag options using lists as arguments may have the list specified in one of two forms: a list of identifiers separated from one another by a comma, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

The flag options are:

-e	Print information about all processes.
-d	Print information about all processes, except process group leaders.
-a	Print information about all processes, except process group leaders and processes not associated with a terminal.
-f	Generate a *full* listing. (See below for meaning of columns in a full listing).
-l	Generate a *long* listing. See below.
-c*corefile*	Use the file *corefile* in place of /dev/kmem.
-s*swapdev*	Use the file *swapdev* in place of /dev/swap. This is useful when examining a *corefile*; a *swapdev* of /dev/null will cause the user block to be zeroed out.
-n*namelist*	The argument will be taken as the name of an alternate *namelist* file in place of /unix.
-t*termlist*	Restrict listing to data about the processes associated with the terminals given in *termlist*. The *termlist* may be in one of two forms: a list of terminal identifiers separated from one another by a comma, or a list of terminal identifiers enclosed in double quotes and separated from one another by a

command and/or one or more spaces. Terminal identifiers may be specified in one of two forms: the device's filename (e.g., `tty04`), or, if the device's filename starts with `tty`, just the digit identifier (e.g., `04`).

−p*proclist* Restrict listing to data about processes whose process ID numbers are given in *proclist*.

−u*uidlist* Restrict listing to data about processes whose user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID will be printed unless the `-f` flag option is used, in which case the login name will be printed.

−g*grplist* Restrict listing to data about processes whose process group leaders are given in *grplist*.

The column headings and the meaning of the columns in a `ps` listing are given below; the letters `f` and `l` indicate the option (*full* or *long*), respectively, that causes the corresponding heading to appear; *all* means that the heading always appears. Note that these two flag options determine only what information is provided for a process; they do *not* determine which processes will be listed.

`F` `(l)`
Flags (hex and additive) associated with the process:

0	swapped;
1	system process;
2	being traced by another process;
4	another tracing flag;
8	process cannot be woken by a signal;
10	in core;
20	locked in memory;
100	process group leader;
200	faulting in page
400	COFF binary
1000	process is using select system call
2000	timing out during sleep
4000	4.2-style job control
8000	restore old mask after signal

`S` `(l)` The state of the process:
- nonexistent;

S sleeping;
R running;
I intermediate (between states);
Z terminated;
T stopped.
O as running on CPU
X waiting for virtual memory

`UID` `(f,l)`
The user ID number of the process owner; the login name is printed under the `-f` flag option.

`PID` `(all)`
The process ID of the process; it is possible to kill a process if you know this datum.

`PPID` `(f,l)`
The process ID of the parent process.

`C` `(f,l)`
Processor utilization for scheduling.

`PRI` `(l)`
The priority of the process; higher numbers mean lower priority.

`NI` `(l)`
Nice value; used in priority computation.

`ADDR` `(l)`
The memory address of the `u-area` (a pointer to the page tables) of the process, if resident; otherwise, the disk address.

`SZ` `(l)`
The size in logical pages of the core image of the process.

`WCHAN` `(l)`
The event for which the process is waiting or sleeping; if blank, the process is running.

`STIME` `(f)`
Starting time of the process.

`TTY` `(all)`
The controlling terminal for the process.

`TIME` `(all)`
The cumulative execution time for the process.

`COMMAND` `(all)`
The command name; the full command name and its arguments are printed under the `-f` flag option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked `defunct`.

Under the `-f` flag option, `ps` tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the `-f` flag option, is printed in square brackets.

EXAMPLES

```
ps -ef
```

displays information about all processes, with or without terminals.

FILES

`/bin/ps`	
`/unix`	A/UX kernel
`/dev/kmem`	memory
`/dev/swap`	the default swap device
`/etc/passwd`	supplies UID information
`/etc/ps_data`	internal data structure
`/dev`	searched to find terminal (`tty`) names

SEE ALSO

`acctcom`(1), `kill`(1), `nice`(1), `pstat`(1), `w`(1).

BUGS

Things can change while `ps` is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

Processes which are swapped onto other than the default swap device (see `swap`(1M) will have some invalid information printed out.

NAME

psdit — convert troff intermediate format to POSTSCRIPT format

SYNOPSIS

psdit [-F *fontdir*] [-p *prologue*] [-o *list*] [*file*]

DESCRIPTION

psdit translates a *file* created by device-independent troff(1) to POSTSCRIPT format for printing on a POSTSCRIPT printer. If no *file* is mentioned, the standard input is used. The POSTSCRIPT file is sent to the standard output. The flag options are

-F *fontdir* Take font information from *fontdir* instead of the default, as described later in this section.

-p *prologue* Use the contents of *prologue* instead of the default POSTSCRIPT prologue, as described later in this section.

-o *list* Print pages whose numbers are given in the comma-separated *list*. The list contains single numbers *n* and ranges *n1–n2*. A missing *n1* means the lowest-numbered page; a missing *n2* means the highest.

Note: The input for psdit should be prepared with the corresponding -Tpsc option of troff, pic, grap, and so forth. eqn should be run with the flags -r576 and -m2 to produce suitable output. pic should be run with the -D flag option and the -T576 flag option to set the correct resolution.

psdit allows for users to cause troff to include arbitrary POSTSCRIPT code in the generated POSTSCRIPT file. psdit recognizes the heretofore undefined % command in the troff intermediate file format to signal the start of raw POSTSCRIPT to be placed "as is" in the the output file. Everthing between (but not including) the percent sign and a line containing a single period (.) will be placed in the generated POSTSCRIPT output. This POSTSCRIPT is not insulated from the troff coordinate system or from the state of the generated POSTSCRIPT. However, two functions are defined in the prologue so that user's may insulate themselves, if they so desire. The PB function (for "picture begin") will perform a POSTSCRIPT save operation, translate the POSTSCRIPT coordinate system to troff's idea of the current po-

sition on the page, and change the scale and orienation of the coordinate system axes to the standard POSTSCRIPT 72 units per inch. The `PE` macro (for "picture end") will end this protected environment.

Several methods may be employed to incorporate included POSTSCRIPT into the `troff` intermediate file. The `.cf` `.sy` and `\! troff` commands may be useful. For example, the following sequence may appear anywhere in `troff` input

```
\!%PB
.cf mypic.ps
\!PE
\!.
```

to include `mypic.ps` as an illustration. This facility is both powerful and useful, but indiscriminate inclusion of poorly behaved POSTSCRIPT code may be dangerous to your document's health.

EXAMPLES

The following command line will format the file `ch.1` using the `troff` text formatting program, translate `troff`'s output into POSTSCRIPT, and then send the POSTSCRIPT output to the appropriate printer.

```
troff -Tpsc -mm ch.1 | psdit | lp -dPigs
```

ENVIRONMENT

`PSLIBDIR`	Pathname of a directory to use instead of `/usr/lib/ps` for `psdit` prologue.

FILES

`/usr/bin/psdit`	
`/usr/lib/font/devpsc/*`	`troff` default description files for POSTSCRIPT virtual device.
`/usr/lib/ps/psdit.pro`	default POSTSCRIPT prologue.

SEE ALSO

`lp`(1), `lpr`(1), `psroff`(1), `troff`(1).

BUGS

The B-splines generated by `troff` are drawn with an approximation. The functions `D~` and `D~~` in the prologue need a little work.

NAME

psroff — troff to a POSTSCRIPT printer

SYNOPSIS

psroff [-t] [*troff-option...*] [*spool-option...*] [*file...*]

DESCRIPTION

psroff is a shell script that runs troff(1) in an environment to produce output on a POSTSCRIPT printer. It uses psdit to convert troff intermediate output to POSTSCRIPT format, and spools this for printing. If no *files* are specified, the standard input is used.

In addition to the standard troff options, the following options are understood by psroff.

-t Send the POSTSCRIPT output to the standard output rather than spooling it to a printer. Note that this overrides the meaning of the troff -t option; if that option is needed, then run troff directly.

The following spooler options are passed on to lp.

-d*dest* Causes the output to be sent to the named destination.

-n*n* Causes *n* copies of the output to be produced. The default is one.

-h Suppresses the printing of the job burst page.

-r Doesn't page-reverse the output.

-s Suppresses messages from lp.

-m Sends mail after files have been printed.

-w Writes to the user's terminal after files have been printed.

Using psroff is equivalent to using the standard pipeline of commands

```
troff -Tpsc options | psdit | lp options
```

Using psroff instead of this pipeline involves less typing, but the entire sequence may take slightly more time since a shell script will be executed.

ENVIRONMENT

LPDEST	The name of a printer (as in the -d option) for lp to use. If no -d option is specified, lp will use this printer. If neither -d nor LPDEST is set, psroff will spool to a printer class named PostScript.

FILES

/usr/bin/psroff	
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/font/devpsc/*	troff description files for POSTSCRIPT virtual device.

SEE ALSO

daiw(1), eqn(1), lpr(1), lp(1), pic(1), psdit(1), refer(1), tbl(1), troff(1).

"nroff/troff Reference" in *A/UX Text Processing Tools.*

BUGS

The eqn supplied with troff is different than the original. Use the options -r576 -m2 for best results. Other programs (for example, pic) distributed with troff have the device names compiled in (so much for device independence!). Use -T576 with the pic distributed with the Documenter's Workbench. If your output is destined for an Apple ImageWriter II printer, use the -Tiw flag option for both pic and eqn.

NAME

ptx — make permuted index

SYNOPSIS

ptx [-b *break*] [-f] [-g *gap*] [-i *ignore*] [-o *only*] [-r] [-t] [-w *n*] [*input* [*output*]]

DESCRIPTION

ptx generates the file *output* that can be processed with a text formatter (nroff or troff) to produce a permuted index of file *input*. Standard input (-) and standard output are default. ptx has three phases: first, the permutation is done, generating one line for each keyword in an input line. The keyword is rotated to the front. Second, the permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of each line. ptx output is in the following form:

.xx "*tail*" "*before keyword*" "*keyword and after*" "*head*"

where .xx is assumed to be an nroff(1) or troff(1) macro provided by the user or provided by ptx(1). The mptx(5) macro package provides the .xx macro definition. The *before keyword* and *keyword and after* fields incorporate as much of the line as fits around the keyword when it is printed. The *tail* and *head* fields, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

FLAG OPTIONS

The following flag options can be applied:

-f	Fold upper and lowercase letters for sorting.
-t	Prepare the output for the phototypesetter.
-w *n*	Use the next argument *n* as the length of the output line. The default line length is 72 characters for nroff and 100 for troff.
-g *n*	Use the next argument *n* as the number of characters that ptx reserves in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3.
-o *only*	Use as keywords any words given in the file *only*. This option cannot be used with the -i option.
-i *ignore*	Do not use as keywords any words given the file *ignore*. If the -i and -o options are missing, use /usr/lib/eign as the file *ignore*. This option cannot be used with the -o option.

`-b` *break* Use the characters in the file *break* to separate words. Tab, newline, and space characters are always used as break characters.

`-r` Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter), separate from the text of the line. Attach that identifier as a 5th field on each output line.

FILES

`/usr/bin/ptx`
`/bin/sort`
`/usr/lib/eign`
`/usr/lib/tmac/tmac.ptx`

SEE ALSO

`troff`(1), `mm`(5), `mptx`(5).

"Other Text Processing Tools," in *A/UX Text Processing Tools.*

BUGS

Line-length counts do not account for overstriking or proportional spacing. Lines that contain tildes (`~`) are botched because `ptx` uses that character internally. `ptx` does not discard nonalphanumeric characters.

NAME

pwd — print working directory name

SYNOPSIS

```
pwd
```

DESCRIPTION

pwd prints the pathname of the working (current) directory.

EXAMPLES

```
pwd
```

produces a pathname, such as /usr/games, indicating the directory you are currently in.

FILES

/bin/pwd

SEE ALSO

csh(1), ksh(1), sh(1).

DIAGNOSTICS

"Cannot open .." and "Read error in ..." indicate possible file system trouble and should be referred to a system administrator.

NAME

query — query the user for input

SYNOPSIS

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

DESCRIPTION

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

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

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

-m
: Watch 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.

DIAGNOSTICS

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

SEE ALSO

line(1).

NAME

rcp — remote file copy

SYNOPSIS

rcp *file1 file2*

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

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 :).

If the -r option is specified and any of the source files are directories, rcp copies each subtree rooted at that name; in this case the destination must be a directory.

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 section "Network Per-

missions" of "Using B-NET" in *A/UX Communications User's Guide* for more information.

FILES

`/usr/bin/rcp`

SEE ALSO

cp(1), `ftp`(1C), `remsh`(1N), `rlogin`(1N).
"Using B-NET" in *A/UX Communications User's Guide.*

BUGS

`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.

NAME

rcs — change RCS file attributes

SYNOPSIS

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

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).

-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.

-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.

-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.

-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.

`-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`.

`-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.

`-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.

`-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.

`-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*]
Same as `-n`, except that it overrides a previous assignment of *name*.

`-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` 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.

DIAGNOSTICS

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.

FILES

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.

DISCLAIMER

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.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN 47907.

SEE ALSO

ci(1), co(1), ident(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rlog(1), rcsfile(4), sccstorcs(1M).

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 — compare RCS revisions

SYNOPSIS

rcsdiff [-biwt] [-cefhn] [-r*rev1*] [-r*rev2*] *files*

DESCRIPTION

rcsdiff runs diff(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.

All options except -r have the same effect as described in diff(1).

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 *rev1* is given, but *rev2* is omitted, rcsdiff compares revision *rev1* of the RCS file with the contents of the corresponding working file.

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

DISCLAIMER

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.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN 47907.

SEE ALSO

ci(1), co(1), diff(1), ident(1), rcs(1), rcsintro(1), rcsmerge(1), rlog(1), rcsfile(4).

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 — merge RCS revisions

SYNOPSIS

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

DESCRIPTION

rcsmerge incorporates the changes between *rev1* and *rev2* of an RCS file into the corresponding working file. If -p is given, the result is printed on the standard output; otherwise the result overwrites the working file.

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

rev1 may not be omitted. If *rev2* is omitted, the latest revision on the trunk is assumed. Both *rev1* and *rev2* may be given numerically or symbolically.

rcsmerge 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.

DISCLAIMER

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.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.

Revision Number: 3.0; Release Date: 83/01/15.

SEE ALSO

`ci`(1), `co`(1), `merge`(1), `ident`(1), `rcs`(1), `rcsdiff`(1), `rlog`(1), `rcsfile`(4).

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.

BUGS

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

NAME

rcvhex — receive and convert Motorola S-records from a port to a file

SYNOPSIS

rcvhex [–p *port*] [–c *command*] *file*

DESCRIPTION

rcvhex translates Motorola S-records shipped from a port into a file. The following flag options are available:

p *port* specifies an alternate port for reception; the default port is /dev/tty0.

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

ifile File to be created by rcvhex.

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

FILES

/usr/bin/rcvhex

SEE ALSO

/usr/bin/hex(1).

NAME

rdist — remote file distribution program

SYNOPSIS

rdist [-nqbRhivwy] [-f*distfile*] [-d*var=value*] [-m*host*] [*name...*]

rdist [-nqbRhivwy] -c *name...* [*login@*] *host* [:*dest*]

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. If the -f flag option is not present, the program looks first for *distfile*, then *Distfile* to use as the input. 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 file names conflict, it is assumed to be a label. These may be used together to update specific files using specific commands.

The -c flag option forces rdist to interpret the remaining arguments as a small *distfile*. The equivalent *distfile* is as follows.

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

Other flag options:

-d Define *var* to have *value*. The -d flag 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.

-m Limit 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 Print the commands without executing them. This flag option is useful for debugging *distfile*.

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

-R Remove extraneous files. If a directory is being updated, any files that exist on the remote host that do not exist in the mas-

ter directory are removed. This is useful for maintaining truely identical copies of directories.

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

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

-v Verify 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 Whole mode. The whole file name 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 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.

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

distfile contains 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 flag option is used with a file name 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 file name.

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. *opt_dest_name* is an optional parameter to rename files. If no install command appears in the command list or the destination name is not specified, the source file name is used. Direc-

tories in the path name 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 flag 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 login name used on the destination host is the same as the local host unless the destination name is of the format "*login@host*".

The command is used to mail the list of files updated (and any errors that may have occured) 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 file name, 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*. *string* starts and ends with " 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.

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} -> ${HOSTS}
        install -R ;
        except /usr/lib/${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 ;
```

FILES

`/usr/bin/rdist`	
distfile	input command file
`/tmp/rdist*`	temporary file for update lists

SEE ALSO

`sh`(1), `csh`(1), `stat`(2).

DIAGNOSTICS

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.

BUGS

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.

See ed(1)

NAME

refer — find and insert literature references in documents

SYNOPSIS

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

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.

FLAG OPTIONS

The following flag options are available.

-a*n* Reverse 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 Bare mode: do not put any flags in text (neither numbers nor labels).

-c*keys*
Capitalize (with CAPS SMALL CAPS) the fields whose keyletters are in *keys*.

-e Instead of leaving the references where encountered, accumulate them until a sequence of the form

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

is encountered, and then write out all references collected so far and collapse the references to the same source.

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

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

`-l`*m*,*n*
Instead of numbering references, use labels made from the senior author's last name and the year of publication. 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` Do 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` flag has no effect.

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

`-s` *keys*
Sort references by fields whose keyletters are in the *keys* string; permute the reference numbers in text accordingly. This flag 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`.

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

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

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

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 keyletters and their corresponding fields, see addbib(1). An example of a refer entry is given below.

EXAMPLES

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

FILES

/usr/ucb/refer	
/usr/dict/papers	Directory of default publication lists
/usr/lib/refer	Directory of companion programs

SEE ALSO

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

BUGS

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.

NAME

regcmp — regular expression compile

SYNOPSIS

regcmp [–] *file* ...

DESCRIPTION

regcmp, in most cases, precludes the need for calling regcmp(3X) from C programs. This saves on both execution time and program size. The command regcmp compiles the regular expressions in *file* and places the output in *file*.i. If the – flag option is used, the output will be placed in *file*.c. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of regcmp is C source code. Compiled regular expressions are represented as extern char vectors. *file*.i files may thus be *included* into C programs, or *file*.c files may be compiled and later loaded. In the C program which uses the regcmp output, *regex*(*abc*,*line*) will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

EXAMPLES

```
name "([A-Za-z][A-Za-z0-9_]*)$0"

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

In the C program that uses the

regcmp *output*,

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

will apply the regular expression named telno to line.

FILES

/usr/bin/regcmp

SEE ALSO

cc(1), lex(1), regcmp(3X).

NAME

remlogin — remote sign on

SYNOPSIS

remlogin

DESCRIPTION

The remlogin command is used when a user initially signs on from a remote host.

When remlogin is invoked from a virtual terminal server process, it asks for a user name, and, if appropriate, a password. Echoing is turned off (if possible) during the typing of the password, so it will not appear on the written record of the session.

After a successful login, accounting files are updated and the user is informed of the existence of mail. The message of the day is printed, as is the time of his last login. Both are suppressed if he has a .hushlogin file in his home directory; this is mostly used to make life easier for non-human users, such as uucp.

remlogin initializes the user and group IDs and the working directory, then executes a command interpreter (usually csh(1)) according to specifications found in a password file. Argument 0 of the command interpreter is the name of the command interpreter with a leading dash (-).

remlogin also modifies the environment environ(7) with information specifying home directory, command interpreter, terminal type (if available) and user name.

If the file /etc/nologin exists, login prints its contents on the user's pseudo terminal and exits. This is used by shutdown(1M) to stop users logging in when the system is about to go down.

FILES

/usr/spool/mail/*	
/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/spool/mail/*	mail
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/nologin	stops logins
.hushlogin	makes login quieter

SEE ALSO

mail(1), passwd(1), getty(1M), rlogind(1M), telnetd(1M), init(1M), shutdown(1M), rlogin(1N), passwd(5), environ(7).

DIAGNOSTICS

`Login incorrect`
: if the name or the password is bad.

`No Shell`
: cannot open password file

`no directory`
: consult a system administrator.

BUGS

`remlogin` uses two undocumented options. `-r` is used by the remote login server, `rlogind`(1M) to force `remlogin` to enter into an initial connection protocol. `-h` is used by `telnetd`(1M) and other servers to list the host from which the connection was received.

NAME

remsh — remote shell

SYNOPSIS

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

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. remsh 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.

The remote user name used is the same as your local user name, unless you specify a different remote name with the -l flag option. 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.

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

If you stipulate the -n flag option, the standard input is redirected to /dev/null.

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 *rhost* 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
/etc/hosts
/usr/hosts/*

SEE ALSO

rlogin(1N), telnet(1N).

See tset(1)

NAME

rev — reverse characters within each line of text

SYNOPSIS

rev [*file*] ...

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

NAME

rez — compile resources

SYNOPSIS

rez [*option*]... [*resource-description-file*]...

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 an appendix of *A/UX Toolbox*.

The *resource-description-file* parameter represents the names of one or more files containing resource descriptions. If no filenames are specified, rez accepts keyboard input.

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 /usr/lib/mac/rincludes.

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

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

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

rez returns one of the following status values.

0 No errors

1 Error in parameters

2 Syntax error in file

3 I/O or program error

You may specify one or more of the following options.

-align[word|longword]
Align resources along word or longword boundaries. This may allow the Resource Manager to load these resources faster. The -align option is ignored when the -a option is in

effect.

-a[ppend]
Output from rez is appended to the output file instead of replacing the output file.

Note: rez overwrites any existing resource of the same type and ID without a warning message. rez cannot append resources to a resource file in which the Read Only bit is set. Also, rez cannot replace a resource file that has a protection bit set. See also the -ov option below.

-c[reator] *creator-expr*
Set the creator of the output file. (The default value is ????.)

-d[efine] *macro*[=*data*]
Define the macro variable *macro* to have the value *data*. If *data* is omitted, then *macro* is set to the null string (this still means that *macro* is defined). The -d option is the same as writing

```
#define macro [data]
```

at the beginning of the input.

-i*pathname(s)*
Search the specified *pathnames* for include files. You may specify more than one pathname. The paths are searched in the order they appear in the command line.

To reach the include files provided with the A/UX Toolbox, use this pathname

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

-o *output-file*
Place the output in *output-file*. Specify the name of the associated data file; rez automatically affixes a percent sign (%) to the name of the header file containing the resources. The default output file is rez.Out.

-ov
Override the protected bit when replacing resources with the -a option.

-p[rogress]
Write version and progress information to diagnostic output.

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

`-ro`
Set the `mapReadOnly` flag in the resource map.

`-s` *pathname(s)*
Search the specified *pathnames* for resource include files.

`-t[ype]` *type-expr*
Set the type of the output file. The default value is `APPL`.

`-u[ndef]` *macro*
Undefine the macro variable *macro*. This is the same as writing

```
#undef macro
```

at the beginning of the input. It is meaningful to undefine the preset macro variables only. (See Appendix C of the document *A/UX Toolbox* for a description of macro variables.)

EXAMPLES

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

generates a resource file for `sample`, based on the descriptions in `sample.r` and the include files in the directory `/usr/lib/mac/rincludes`. Place the output in an AppleDouble header file named `%sample`.

FILES

`/mac/bin/rez`

SEE ALSO

`derez`(1).

NAME

rlog — display log messages and other information about RCS files

SYNOPSIS

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

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).

rlog 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 below restrict this output.

FLAG OPTIONS

The following flag options are interpreted by rlog:

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

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

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

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

-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.

-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.

-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.

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

-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.

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

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 sub-

directory 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.

DIAGNOSTICS

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

DISCLAIMER

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 and for use at your own discretion.

The source code is normally found in /usr/src or is made available through the Apple Programmer's and Developer's Association (APDA™). This source code should also be used at your own risk and without support from Apple.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN 47907.

SEE ALSO

ci(1), co(1), ident(1), rcs(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rcsfile(4), sccstorcs(1M).

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 — remote login

SYNOPSIS

rlogin *rhost* [-8] [-ec] [-l *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 *rhosts* 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.

The -8 flag option allows an eight-bit data path; otherwise parity bits are stripped.

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.

A different escape character can be specified with the -e flag option:

```
rlogin -ex
```

defines the character "x" as the escape character, with no space separating the option and the argument character.

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.

FILES

`/usr/bin/rlogin`	
`/usr/hosts/*`	for *rhost* version of the command

SEE ALSO

`remsh`(1N), `stty`(1).

BUGS

More terminal characteristics should be propagated.

NAME

rm, rmdir — remove files or directories

SYNOPSIS

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

rmdir *dir...*

DESCRIPTION

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

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with y, the file is deleted, otherwise the file remains. No questions are asked when the -f flag option is given or if the standard input is not a terminal. The -f flag option also prevents all error messages from being printed.

If a designated file is a directory, an error comment is printed unless the optional argument -r has been used. In that case, rm recursively deletes the entire contents of the specified directory, and the directory itself.

If the -i (interactive) flag option is in effect, rm asks whether to delete each file, and, under -r, whether to examine each directory.

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

EXAMPLES

```
rm -rf dirname
```

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

FILES

/bin/rm
/bin/rmdir

SEE ALSO

mkdir(1), unlink(2).

DIAGNOSTICS

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

```
rm -r .*
```

See mail(1)

NAME

rmdel — remove a delta from an SCCS file

SYNOPSIS

rmdel -r *SID file* ...

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

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

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

FILES

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

SEE ALSO

admin(1), cdc(1), comb(1), delta(1), get(1), help(1), prs(1), remdel(1), sact(1), sccsdiff(1), unget(1), val(1), what(1), sccsfile(4).
"SCCS Reference" in *A/UX Programming Languages and Tools, Volume 2*.

DIAGNOSTICS

Use help(1) for explanations.

See rm(1)

NAME

roffbib — run off bibliographic database

SYNOPSIS

roffbib [-e] [-h] [-n] [-o] [-r] [-s] [-T*term*] [-x] [-m *mac*] [-V] [-Q] [*file*...]

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
```

roffbib accepts most of the flag options understood by nroff(1); 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. The -x flag option will suppress the printing of these abstracts.

A user-defined set of macros may be specified after the -m flag option. There should be a space between the -m and the macro filename. This set of macros will replace the ones defined in /usr/lib/tmac/tmac.bib. The -V flag option will send output to the Versatec printer/plotter; the -Q flag option will queue output for the phototypesetter.

Four command-line registers control formatting style of the bibliography, much like the number registers of ms(5). 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).

Note: with the -V and -Q flags, the default page offset is already one inch.

FILES

/usr/ucb/roffbib
/usr/lib/tmac/tmac.bib

SEE ALSO

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

BUGS

Users have to rewrite macros to create customized formats.

See sh(1)

NAME

rup — show status of machines on local network (RPC version)

SYNOPSIS

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

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 listed below.

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).

FLAG OPTIONS

-h sort the display alphabetically by host name.

-l sort the display by load average

-t sort the display by up time.

FILES

/usr/bin/rup
/etc/servers

SEE ALSO

inetd(1M), rstatd(1M), ruptime(1N).

BUGS

Broadcasting does not work through gateways.

NAME

ruptime — show host status of local machines

SYNOPSIS

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

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.

Users idle an hour or more are not counted unless the -a flag is given.

Normally, the listing is sorted by host name. The -l, -t, and -u flags specify sorting by load average, uptime, and number of users, respectively.

FILES

/usr/bin/ruptime
/usr/spool/rwho/whod.*

SEE ALSO

rwho(1N), uptime(1).

NAME

rusers — give login list for local machines (RPC version)

SYNOPSIS

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

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. When *host* arguments are given, rusers will query only the list of specified hosts, rather than broadcasting to the entire network.

The default is to print out a listing in the style of users(1) with one line per machine. When the -l flag option is given, a rwho(1) style listing is used. In addition, if a user hasn't typed to the system for a minute or more, the idle time is reported.

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

FLAG OPTIONS

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

-h Sorts alphabetically by host name.

-i Sorts by idle time.

-l Gives a longer listing in the style of who(1).

-u Sorts by number of users.

FILES

/usr/etc/rusers
/etc/servers

SEE ALSO

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

NAME

rwho — who's logged in on local machines?

SYNOPSIS

```
rwho [-a]
```

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` flag is given.

FILES

```
/usr/bin/rwho
/usr/spool/rwho/whod.*
```

SEE ALSO

`ruptime`(1N), `rwhod`(1M).

BUGS

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

NAME

sact — display who has checked an SCCS file out for editing.

SYNOPSIS

sact *file* ...

DESCRIPTION

sact informs the user of any impending deltas to a named SCCS file. This situation occurs when get(1) with the -e flag option has been previously executed without a subsequent execution of delta(1). If a directory is named on the command line, sact behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

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

Field 1 specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.

Field 2 specifies the SID for the new delta to be created.

Field 3 contains the logname of the user who will make the delta (i.e., executed a 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 done a get -e, but not a delta to merge the new changes, doing a

```
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 for the file was done on 82/11/10 at 16:10:35.

FILES

`/usr/bin/sact`

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

DIAGNOSTICS

Use `help`(1) for explanations.

NAME

sag — system activity graph

SYNOPSIS

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

DESCRIPTION

sag graphically displays the system activity data stored in a binary data file by a previous sar(1) run. Any of the sar data items may be plotted singly, or in combination; as cross plots, or versus time. Simple arithmetic combinations of data may be specified. sag invokes sar and finds the desired data by string-matching the data column header (run sar to see what's available). These flag options are passed through to sar:

-e *time* Select 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. Default is 18:00.

-f *file* Use *file* as the data source for sar. Default is the current daily data file /usr/adm/sa/sa*dd*.

-i *sec* Select data at intervals as close as possible to *sec* seconds.

-s *time* Select data later than *time*. Default is 08:00.

Other flag options:

-T*term* Produce 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* x axis specification with *spec* in the form:

name [*op name*] . . . [*lo hi*]

-y *spec* y axis specification with *spec* in the same form as above.

name is either a string that will match a column header in the sar report, with an optional device name in square brackets, e.g., r+w/s[dsk-1], or an integer value. *op* is +, –, *, or / surrounded by blanks. Up to five names may be specified. Parentheses are not recognized. Contrary to custom, + and – have precedence over * and / . Evaluation is left to right. Thus

```
A / A + B * 100
```

is evaluated as

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

and

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

is

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

lo and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

A single *spec* is permitted for the x axis. If unspecified, *time* is used. Up to 5 *spec*'s separated by ; may be given for –y. Enclose the –x and –y arguments in " " if blanks or \RETURN are included. The –y default is:

```
–y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"
```

EXAMPLES

```
sag
```

will show today's CPU utilization.

FILES

`/usr/bin/sag`	
`/usr/adm/sa/sa`*dd*	daily data file for day *dd*.

SEE ALSO

sar(1), tplot(1G).

NAME

sar — system activity reporter

SYNOPSIS

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

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

DESCRIPTION

sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the -o flag option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, sar extracts data from a previously-recorded *file*, either the one specified by the -f flag option or, by default, the standard system activity daily data file /usr/adm/sa/sa*dd* for the current day *dd*. The starting and ending times of the report can be bounded via the -s and -e *time* arguments of the form *hh*[:*mm*[:*ss*]]. The -i option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by one of the following options:

-u Report CPU utilization (the default):
%usr, %sys, %wio, %idle: portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.

-b Report buffer activity:
bread/s, bwrit/s: transfers per second of data between system buffers and disk or other block devices.
lread/s, lwrit/s: accesses of system buffers.
%rcache, %wcache: cache hit ratios, for example, 1-bread/lread.
pread/s, pwrit/s: transfers via raw (physical) device mechanism.

-y Report 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.

-c Report system calls.
`scall/s`: system calls of all types.
`sread/s`, `swrit/s`, `fork/s`, `exec/s`: specific system calls.
`rchar/s`, `wchar/s`: characters transferred by `read` and `write` system calls.

-w Report 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.

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

-q Report 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.

-v Report 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.

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

-A Report all data. Equivalent to `-uqbwcayvm`.

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`

`/usr/adm/sa/sa`*dd*	daily data file, where *dd* are digits representing the day of the month

SEE ALSO

`sag`(1G), `sadc`(1M).

"System Activity Package" in *A/UX Local System Administration*.

NAME

sccs — front end for the SCCS subsystem

SYNOPSIS

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

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 *command* with the specified *flags* and *args*. Each argument is normally modified to be prefixed with SCCS/s.. Thus, you may run get, delta, or info as such a *command*.

Flags to be interpreted by the sccs program must appear before the *command* argument. Flags to be passed to the actual SCCS program must come after the *command* argument. These flags are specific to the command and are discussed in the documentation for that command.

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

edit	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, and -y flags will be passed to delta, and the -b, -c, -e, -i, -k, -l, -s, and -x flags will be passed to get.
deledit	Equivalent to delget, except that the get phase includes the -e flag. This option is useful for making a *checkpoint* of your current editing phase. The same flags will be passed to delta as described above, and all the flags listed for get above except -eand-k are passed to edit.
create	Create an SCCS file, taking the initial contents from the file of the same name. Any flags 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` flag. 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.

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` flag 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` flag is given, branches (i.e., SID's with two or fewer components) are ignored. If the `-u` flag 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 insure 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` flags 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` flags are passed to `get`; the `-l`, `-s`, `-e`, `-f`, `-h`, and `-b` options are passed to

diff. The -C flag is passed to diff as -c.

prs — This command prints out verbose information about the named files.

The -r flag runs sccs as the real user rather than as whatever effective user sccs has set user ID to. The -d flag gives a root directory for the SCCS files. The default is the current directory. The -p flag defines the pathname of the directory in which the SCCS files will be found; SCCS is the default. The -p flag differs from the -d flag in that the -d argument 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 flag. 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.

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) $@
```

FILES

`/usr/ucb/sccs`

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

BUGS

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

NAME

sccsdiff — compare two versions of an SCCS file

SYNOPSIS

sccsdiff −r*SID1* −r*SID2* [−p] [−s*n*] *file...*

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.

−r*SID?* *SID* and *SID* specify the deltas of an SCCS file that are to be compared. Versions are passed to bdiff(1) in the order given.

−p pipe output for each file through pr(1).

−s*n* *n* is the file segment size that bdiff will pass to diff(1). This is useful when diff fails due to a high system load.

EXAMPLES

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

would show the differences between version 1.1 and version 1.2 of the file test1.c.

FILES

/usr/bin/sccsdiff
/tmp/get?????

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 the *A/UX Programming Languages and Tools, Volume 2*.

DIAGNOSTICS

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

Use help(1) for explanations.

NAME

script — start a shell that records terminal input and output

SYNOPSIS

script [-a] [*file*]

DESCRIPTION

script makes a typescript of everything printed on your terminal. The typescript is written to *file*, or appended to *file* if the -a flag option is given. It can be sent to the line printer later with or lpr(1). If no file name is given, the typescript is saved in the file typescript.

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.

FILES

/usr/bin/script

SEE ALSO

lpr(1).

BUGS

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.

NAME

sdb — symbolic debugger

SYNOPSIS

sdb [–w] [–W] [*objfil* [*corfil* [*directory*]]]

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.

objfil is normally an executable program file which has been compiled with the –g (debug) flag option; if it has not been compiled with the –g flag 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 *objfil* is a.out. *corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is core. The core file need not be present. A – in place of *corfil* forces sdb to ignore any core image file. Source files used in constructing *objfil* must be in *directory* to be located.

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

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

Names of variables are written just as they are in C or f77(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 *filename*: *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 *objfil* or *corfil*. An initial argument of `-w` permits overwriting locations in *objfil*.

Addresses

The address in a file associated with a written address is determined by a mapping associated with that file. 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:

b1≤*address*<*e1*

file address=*address*+*f1*−*b1*

otherwise

b2≤*address*<*e2*

file address=*address*+*f2*−*b2*,

otherwise, the requested *address* is not legal. In some cases (e.g., 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` Print a stack trace of the terminated or halted program.

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

variable`/`*clm*

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

- `b` 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` Assume *variable* is a string pointer and print characters starting at the address pointed to by the variable.
- `a` Print characters starting at the variable's address. This format may not be used with register variables.
- `p` pointer to procedure
- `i` Disassemble machine-language instruction with addresses printed numerically and symbolically.
- `I` Disassemble machine-language instruction with addresses 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*
Print the value at the address from a.out or I space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is 'i'.

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

variable!*value*
Set *variable* to the given *value*. The value may be a number, a character constant or a variable. The value must be well defined; expressions 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 Print the machine registers and the current machine-language instruction.

X Print the current machine-language instruction.

The commands for examining source files are:

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

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

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

p
: Print the current line.

z
: Print the current line followed by the next 9 lines. Set the current line to the last line printed.

w
: Window. Print the 10 lines around the current line.

number
: Set the current line to the given line number. Print the new current line.

count+
: Advance the current line by *count* lines. Print the new current line.

count -
: Retreat the current line by *count* lines. Print the new current line.

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

count r *args*
count R

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

linenumber c *count*
linenumber C *count*

Continue after a breakpoint or interrupt. If *count* is given, it specifies the 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*

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

s *count*
S *count*

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

i
I

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

variable$m *count*
address:m *count*

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

level v

Toggle verbose mode, for use when single stepping with S,

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

k Kill the program being debugged.

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

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

B Print a list of the currently active breakpoints.

linenumber d
Delete a breakpoint at the given line. If no *linenumber* is given, 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 Delete all breakpoints.

l Print the last executed line.

linenumber a
Announce. If *linenumber* is of the form *proc*:*number*, the

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

Miscellaneous commands:

`!` *command*
The command is interpreted by `sh`(1).

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

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

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

`M` Print the address maps.

`M[/][*]` *b e f*
Record new values for the address map. 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*
Print the given string. The C escape sequences of the form are recognized, where *character* is a nonnumeric character.

`q` Exit the debugger.

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

`V` Print the version number.
`Q` Print a list of procedures and files being debugged.

`Y` Toggle debug output.

FILES

`/usr/bin/sdb`
`a.out`
`core`

SEE ALSO

`adb`(1), `cc`(1), `ctrace`(1), `f77`(1), `sh`(1), `a.out`(4), `core`(4). "`sdb` Reference" in *A/UX Programming Languages and Tools, Volume 1*.

WARNINGS

Data stored in text sections are indistinguishable from functions.

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

BUGS

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

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

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

The range of an `f77` array subscript is assumed to be *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.

NAME

sdiff — side-by-side difference program

SYNOPSIS

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

DESCRIPTION

sdiff uses the output of diff(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

The following flag options exist:

-w *n* Use the next argument, *n*, as the width of the output line. The default line length is 130 characters. The width must be between 20 and 200.

-l Only print the left side of any lines that are identical.

-s Do not print identical lines.

-o *output* Use the next argument, *output*, as the name of a third file that is created as a user controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by diff(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, sdiff prompts the user with a % and waits for one of the following user-typed commands:

l	append the left column to the output file
r	append the right column to the output file
s	turn on silent mode; do not print identical lines
v	turn off silent mode
e l	call the editor with the left column
e r	call the editor with the right column
e b	call the editor with the concatenation of left and right
e	call the editor with a zero length file
q	exit from the program

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

EXAMPLES

If `file1` contains:

```
x
a
b
c
d
```

and `file2` contains:

```
y
a
d
c
```

then

```
sdiff file1 file2
```

would print:

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

FILES

`/usr/bin/sdiff`

SEE ALSO

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

NAME

sed — stream editor

SYNOPSIS

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

sed [-n] -f *sfile* [*file...*]

DESCRIPTION

sed copies the named *files* (standard input default) to the standard output, edited according to a script of sed commands. The -f flag option causes the script to be taken from file *sfile*. The -e flag option causes the script to be taken directly from the command line. These flag options accumulate, so many scripts can be used in one invocation of the command. If there is just one -e flag option and no -f flag options, the -e flag 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 flag option.

The -n flag option 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).

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.

An *address* is either a decimal number that counts input lines cumulatively across files, a $ that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of ed(1) modified 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.

The *text* argument used in some of the commands consists of 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.

An *rfile* or *wfile* argument in a command must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1)`a\`
text
: Append. Place *text* on the output before reading the next input line.

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

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

(2)`d` Delete the pattern space. Start the next cycle.

(2)`D` Delete the initial segment of the pattern space through the first *newline*. Start the next cycle.

(2)`g` Replace the contents of the pattern space by the contents of the hold space.

(2)`G` Append the contents of the hold space to the pattern space.

(2)`h` Replace the contents of the hold space by the contents of the pattern space.

(2)`H` Append the contents of the pattern space to the hold space.

(1)`i\`
text Insert. Place *text* on the standard output.

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

(2)`n` Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

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

(2)`p` Print. Copy the pattern space to the standard output.

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

(1)`q` Quit. Branch to the end of the script. Do not start a new cycle.

(1)`r` *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

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

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

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

`p` Print the pattern space if a replacement was made.

`w` *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2)`t` *label* Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a `t`. If *label* is empty, branch to the end of the script.

(2)`w` *wfile* Write. Append the pattern space to *wfile*.

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

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

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

(0) `:` *label* This command does nothing; it bears a *label* for `b` and `t` commands to branch to.

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

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

(0) An empty command is ignored.

(0)# If a # appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the # is an "`n`", then the default output will be suppressed, as if the –`n` flag 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 byt the first line of the file. A script file must contain at least one non-comment 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.

FILES

`/bin/sed`

SEE ALSO

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

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.

NAME

settc — set the type and creator of a Macintosh resource file

SYNOPSIS

settc *type creator* [*file*]...

DESCRIPTION

settc 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.

The *type* and *creator* parameters are the actual strings used by a Macintosh application to identify itself and the files it creates. If the *type* or *creator* code includes spaces, tabs, or metacharacters, enclose the code in quotation marks.

The *file* parameter represents the name of the 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. settc automatically looks for the associated header file, which should have the same name as the data file, but with a percent sign (%) prefixed. If settc cannot find the header file, it creates one.

Most Macintosh applications open a document file only if they recognize the type and creator. In the Macintosh OS, a file's type and creator are stored in the directory. In A/UX, the type and creator are stored 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 using the normal File Manager routines, it automatically creates the an AppleSingle file with the appropriate type and creator (see the description of the File Manager in Chapter 4 of *Inside Macintosh*).

settc is useful in two situations.

1. When a file's type and creator are lost during a file transfer from the Macintosh environment to A/UX.
2. 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

The command

```
settc PNTG MPNT report house
```

establishes the type of the data files `report` and `house` as `PNTG` and the creator as `MPNT`.

FILES

`/mac/bin/settc`

SEE ALSO

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

NAME

sh, rsh — run the Bourne shell, the earliest of the command interpreters available

SYNOPSIS

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

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

DESCRIPTION

sh is a command programming language that executes commands read from a terminal or a file. rsh is a restricted version of the standard command interpreter sh; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See "Invocation" below for the meaning of arguments to the shell.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters: *, @, #, ?, -, $, and !.

Commands

A *simple-command* is a sequence of 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 ;, &, &&, or ||, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that

of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol && (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (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`
: Each time a `for` command is executed, *name* is set to the next *word* taken from the `in` *word* list. If `in` *word* ... is omitted, then the `for` command executes the `do` *list* once for each positional parameter that is set (see "Parameter Substitution," below). Execution ends when there are no more words in the list.

`case` *word* `in` [*pattern* [| *pattern*]...) *list* ;;]...`esac`
: A `case` command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see "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`
: The *list* following `if` is executed and, if it returns a zero exit status, the *list* following the first `then` is executed. Otherwise, the *list* following `elif` is executed and, if its value is zero, the *list* following the next `then` is executed. Failing that, the `else` *list* is executed. If no `else` *list* or `then` *list* is executed, then the `if` command returns a zero exit status.

`while` *list* `do` *list* `done`
: A `while` command repeatedly executes the `while` *list* and, if the exit status of the last command in the list is zero, executes the `do` *list*; otherwise the loop terminates. If no commands in the `do` *list* are executed, then the `while` command returns a zero exit status; `until` may be used in place of `while` to negate the loop termination test.

(*list*)
: Execute *list* in a subshell.

{ *list* }
list is simply executed.

name () { *list* }
Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see "Execution," 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 to in this way.

${*parameter* :?*word*}
If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "`parameter null or not set`" is printed.

${*parameter* :+*word*}
If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, `pwd` is executed only if `d` is not set or is null:

```
echo ${d:- `pwd`}
```

If the colon (`:`) is omitted from the above expressions, the shell 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 `$`. `$*` 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 in-

voked command; substitution occurs before *word* or *digit* is used:

<*word*	Use file *word* as standard input (file descriptor 0).
>*word*	Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise, it is truncated to zero length.
>>*word*	Use file *word* as standard output. If the file exists, output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
<<[–] *word*	The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) *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> xxx 2>&1
```

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

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

Redirection of output is not allowed in the restricted shell.

Environment

The *environment* (see `environ`(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the 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` flag 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` flag 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      #now prints only 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 `execs` 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 *name*s are marked for automatic export to the *environment* of subsequently-executed commands. If no argu-

ments 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` flag 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 *names* are marked *readonly* and the values of the these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

`return` [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

`set` [`--aefhkntuvx` [*arg* ...]]

`-a` Mark variables which are modified or created for export.

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

`-f` Disable filename generation.

`-h` Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).

`-k` All keyword arguments are placed in the environment for a command, not just those that precede the command name.

`-n` Read commands, but do not execute them.

`-t` Exit after reading and executing one command.

`-u` Treat unset variables as an error when substituting.

`-v` Print shell input lines as they are read.

`-x` Print commands and their arguments as they are executed.

`--` Do not change any of the flags; useful in setting `$1` to `-`.

Using + rather than - causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in `$-`. The remaining arguments are positional parameters and are assigned, in order, to `$1`, `$2`, ... If no arguments are given, the values of all names are printed.

`shift [`*n*`]`
The positional parameters from `$n+1 ...` are renamed `$1 ....` If *n* is not given, it is assumed to be 1.

`test`
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*	true if *file* exists and is readable.
`-w` *file*	true if *file* exists and is writable.
`-x` *file*	true if *file* exists and is executable.
`-f` *file*	true if *file* exists and is a regular file.
`-d` *file*	true if *file* exists and is a directory.
`-c` *file*	true if *file* exists and is a character special file.
`-b` *file*	true if *file* exists and is a block special file.
`-p` *file*	true if *file* exists and is a named pipe (FIFO).
`-u` *file*	true if *file* exists and its set user ID bit is set.
`-g` *file*	true if *file* exists and its set group ID bit is set.
`-k` *file*	true if *file* exists and its sticky bit is set.
`-s` *file*	true if *file* exists and has a size greater than zero.
`-t` [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
`-z` *s1*	true if the length of string *s1* is zero.
`-n` *s1*	true if the length of the string *s1* is 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.

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*. `-f` imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed. If no flag option is given, `-f` is assumed.

`umask`[*nnn*]
The user file-creation mask is set to *nnn* (see `umask`(2)). If *nnn* is omitted, the current value of the mask is printed.

`unset`[*name* ...]
For each *name*, remove the corresponding variable or function. The variables `PATH`, `PS1`, `PS2`, `MAILCHECK`, and `IFS` cannot be unset.

`wait` [*n*]
Wait for the specified process and report its termination status. If *n* is not given, all currently active child processes are waited for and the return code is zero.

Invocation

If the shell is invoked through `exec`(2) and the first character of argument zero is -, commands are 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 flags below are interpreted by the shell on invocation only; note that, unless the `-c` or `-s` flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

`-c` *string* If the `-c` flag is present, commands are read from *string*.

`-s` If the `-s` flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for special commands, see "Special Commands," above) is written to file descriptor 2.

`-i` If the `-i` flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case, Terminate is ignored (so that `kill 0` does not kill an interactive shell) and Interrupt is caught and ignored (so that `wait` is interruptible). In all cases, Quit is ignored by the shell.

`-r` If the `-r` flag is present, the shell is a restricted shell.

The remaining flags and arguments are described under the `set` command above.

`rsh` Only

`rsh` is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of `rsh` are identical to those of `sh`, except that the following are disallowed:

changing directory
setting the value of `$PATH`
specifying path or command names containing `/`
redirecting output (> and >>)

The restrictions above are enforced after `.profile` is interpreted.

When a command to be executed is found to be a shell procedure, `rsh` invokes `sh` to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the `.profile` has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., `/usr/rbin`) that can be safely invoked by `rsh`. Some systems also provide a restricted editor `red`.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a 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).

EXAMPLES

```
sh -x script1
```

will execute each command in `script1`, echoing the command just before executing it.

FILES

```
/bin/sh
/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null
```

SEE ALSO

`acctcom`(1), `csh`(1), `chsh`(1), `echo`(1), `env`(1), `ksh`(1), `login`(1), `newgrp`(1), `pwd`(1), `acctcms`(1M), `exec`(2), `fork`(2), `pipe`(2), `ulimit`(2), `umask`(2), `wait`(2), `dup`(3), `signal`(3), `a.out`(4), `profile`(4), `environ`(5), "Bourne Shell Reference" in *A/UX User Interface*.

CAVEATS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to `exec` the original command. Use the `hash` command to correct this situation.

If you move the current directory or one above it, `pwd` may not give the correct response. Use the `cd` command with a full pathname to correct this situation.

BUGS

Filename pattern matching is not done on redirected I/O filenames.

NAME

shl — shell layer manager

SYNOPSIS

shl

DESCRIPTION

shl allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as *layers*, using the commands described below.

The *current layer* is the layer which can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal.

The stty character swtch (set to CONTROL-z if NULL) is used to switch control to shl from a layer. shl has its own prompt, >>>, to help distinguish it from a layer.

A *layer* is a shell which has been bound to a virtual tty device (/dev/sxt/???). The virtual device can be manipulated like a real tty device using stty(1) and ioctl(2). Each layer has its own process group ID.

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*]

Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form *(#)* where # is the last digit of the virtual device bound to the layer. The shell prompt variable PS1 is set to the name of the layer followed by a space. A maximum of seven layers can be created.

`block` *name* [*name* ...]
For each *name*, block the output of the corresponding layer when it is not the current layer.

`delete` *name* [*name* ...]
For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the `SIGHUP` signal (see `signal`(3)).

`help` (or `?`)
Print the syntax of the `shl` commands.

`layers` [`-l`] [*name* ...]
For each *name*, list the layer name and its process group. The `-l` flag option produces a `ps`(1)-like listing. If no arguments are given, information is presented for all existing layers.

`resume` [*name*]
(followed by RETURN). Make 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). Resume the layer that was current before the last current layer.

`unblock` *name [name* ...]
For each *name*, do not block the output of the corresponding layer when it is not the current layer.

`quit`
Exit `shl`. All layers are sent the `SIGHUP` signal.

name
(followed by RETURN). Make the layer referenced by *name* the current layer.

FILES

`/usr/bin/shl`	
`/dev/sxt/???`	Virtual tty devices
`$SHELL`	Variable containing path name of the shell to use (default is `/bin/sh`) .

SEE ALSO

`sh`(1), `stty`(1), `ioctl`(2), `signal`(3), `sxt`(7).
AIUX User Interface.

NAME

size — display section sizes of common object files

SYNOPSIS

size [-d] [-o] [-V] [-x] *file...*

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.

Numbers are displayed in hexadecimal unless either the -o or the -d flag option is used, in which case they are displayed in octal or in decimal, respectively. If necessary (for example, in a shell script), the -x flag option may be specified to force hexadecimal output.

The -V flag supplies the version information on the size command.

FILES

/bin/size

SEE ALSO

as(1), cc(1), ld(1), a.out(4).

DIAGNOSTICS

size: *name*: cannot open	*name* cannot be read.
size: *name*: bad magic	*name* is not an object file.

NAME

sleep — suspend execution for an interval

SYNOPSIS

sleep *time*

DESCRIPTION

sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

EXAMPLES

label:

```
command >> x
command >> x
date >> x
sleep 10
goto label
```

The preceding sh(1) script would execute the two commands and append the results to file x, then sleep for 10 seconds and repeat the process.

FILES

/bin/sleep

SEE ALSO

alarm(2), sleep(3C).

BUGS

time must be less than 65536 seconds.

NAME

sno — SNOBOL interpreter

SYNOPSIS

sno [*file...*]

DESCRIPTION

sno is a SNOBOL compiler and interpreter (with slight differences). sno obtains input from the concatenation of the named *files* and the standard input. All input through a statement containing the label end is considered program and is compiled. The rest is available to syspit.

sno differs from SNOBOL in the following ways:

There are no unanchored searches. To get the same effect:

*a**b*	unanchored search for *b*.
*a*x*b=xc*	unanchored assignment

There is no back referencing.

x=abc

*a*x*x*	is an unanchored search for abc.

Function declaration is done at compile time by the use of the (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. 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`

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 — eliminate .so's from nroff input

SYNOPSIS

soelim [*file...*]

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 ., for example

```
'so /usr/lib/tmac.s
```

A sample usage of soelim would be:

```
soelim exum?.n | tbl | nroff -mm | col | lp
```

FILES

/usr/ucb/soelim

SEE ALSO

col(1), eqn(1), nroff(1), tbl(1), troff(1),

BUGS

The format of the source commands must be consistent; exactly one blank must precede and no blanks follow the file name.

NAME

sort — sort or merge files

SYNOPSIS

sort [-c] [-m] [-u] [-o *output*] [-y[*kmem*]] [-z*recsz*] [-d] [-f] [-i] [-M] [-n] [-r] [-b] [-t *x*] [+*pos1* [-*pos2*]] [*file...*]

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.

FLAG OPTIONS

The following flag options alter the default behavior:

-c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

-m Merge only because the input files are already sorted.

-u Suppress all but one (unique) line in each set of lines having equal keys.

-o *output*
Place 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*.

-y*kmem*
Sort 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 flag option is omitted, sort begins using a system default memory size and continues to use more space as needed. If this flag 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*
Record 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 flag options, a popular system default size is used. Lines longer than the buffer size causes sort to terminate abnormally. Supplying the actual number of bytes (or some larger value) in the longest line to be merged prevents abnormal termination.

The following flag options override the default ordering rules:

-d Use "dictionary" order. Only letters, digits, and blanks (spaces and tabs) are significant in comparisons.

-f Fold lowercase letters into uppercase.

-i Ignore characters outside the ASCII range 040–0176 in non-numeric comparisons.

-M Compare 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 flag option implies the -b flag option (see later in this section).

-n Sort 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 flag option implies the -b flag option (as described later). Note that the -b flag option is only effective when restricted sort-key specifications are in effect.

-r Reverse the sense of comparisons.

When ordering flag 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 flag options override all global ordering flag options for that key.

The notation +*pos1* -*pos2* restricts a sort key to one beginning at *pos1* and ending just before *pos2*. The characters at 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.

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 flag options.

`-b` Ignore leading blanks when determining the beginning and ending positions of a restricted sort key. If the `-b` flag option is specified before the first *+pos1* argument, it is applied to all *+pos1* arguments. Otherwise, the `b` flag may be attached independently to each *+pos1* or *-pos2* argument (as shown later).

`-t`*x* Use *x* as the field-separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant; for example, *xx* delimits an empty field.

Both of the arguments, *pos1* and *pos2*, have the form *m*.*n* optionally followed by one or more of the flags `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` flag 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.

A last position specified by *-m*.*n* is interpreted to mean the *n*th character (including separators) after the last character of the *m*th field. A missing *.n* means .0, indicating the last character of the *m*th field. If the `b` flag is in effect, *n* is counted from the last leading blank in the *m*+1st field; *-m*.*1b* refers to the first 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 (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 flag options -um, with just one input file, make the choice of a unique representative from a set of equal lines predictable), use the command

```
sort -um +2 -3 infile
```

FILES

/bin/sort
/usr/tmp/stm???

SEE ALSO

comm(1), join(1), rev(1), sortbib(1), tsort(1), uniq(1).

DIAGNOSTICS

sort 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 flag option.

When a newline character is missing from the last line of an input file, sort appends one, prints a warning message, and continues.

NAME

sortbib — sort bibliographic database

SYNOPSIS

sortbib [-s*keys*] *database...*

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.

By default, sortbib alphabetizes by the first %A and the %D fields, which contain the senior author and date. The -s flag option is used to specify new *keys*. 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. No more than 16 databases may be sorted together at one time. Records longer than 4096 characters will be truncated.

sortbib 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.

FILES

/usr/ucb/sortbib

SEE ALSO

addbib(1), indxbib(1), lookbib(1), refer(1), roffbib(1).

BUGS

Records with missing author fields should probably be sorted by title.

NAME

spell, hashmake, spellin, hashcheck — find spelling errors

SYNOPSIS

spell [–v] [–b] [–x] [–l] [+local-file] [*file...*]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin *n*

/usr/lib/spell/hashcheck *spelling-list*

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.

spell ignores most troff(1), tbl(1), and eqn(1) constructions.

Under the –v flag option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the –b flag option, British spelling is checked. In addition to preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, and so on, this flag option insists upon *-ise* in words like *standardise*

Under the –x flag option, every plausible stem is printed with = for each word.

By default, spell, like deroff(1), follows chains of included files (.so and .nx troff(1) requests), *unless* the names of such included files begin with /usr/lib. Under the –l flag option, spell follows the chains of *all* included files.

Under the +*local-file* flag option, words found in *local-file* are removed from the output of spell. The placeholder *local-file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to the spelling list included in spell) for each job.

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.

Three routines help maintain and check the hash lists used by `spell`:

`hashmake` Read a list of words from the standard input and write the corresponding nine-digit hash code on the standard output.

`spellin` *n* Read *n* hash codes from the standard input and write 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` Read a compressed *spelling-list* and recreate the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

EXAMPLES

```
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
```

FILES

```
/bin/spell
/usr/lib/spell
/usr/lib/spell/spellin
/usr/lib/spell/hashcheck
/usr/lib/spell/hashmake
```

```
D_SPELL=/usr/lib/spell/hlist[ab]
S_SPELL=/usr/lib/spell/hstop
H_SPELL=/usr/lib/spell/spellhist
/usr/lib/spell/spellprog
/usr/lib/spell/compress
```

SEE ALSO

diction(1), deroff(1), eqn(1), sed(1), sort(1), style(1), tbl(1), tee(1), troff(1).

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions. Typically, these are kept in a separate local file that is added to the hashed *spelling-list* via spellin.

The British spellings are incomplete.

See spell(1)

NAME

spline — interpolate smooth curve

SYNOPSIS

spline [-a] [-k] [-n] [-p] [-x]

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.

The following flag options are recognized, each as a separate argument:

-a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

-k The constant k used in the boundary value computation:

$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$

is set by the next argument (default $k = 0$).

-n Space output points so that approximately n intervals occur between the lower and upper x limits (default $n = 100$).

-p Make output periodic, i.e., match derivatives at ends. First and last input values should normally agree.

-x Next 1 (or 2) arguments are lower (and upper) x limits. Normally, these limits are calculated from the data. Automatic abscissas start at lower limit (default 0).

EXAMPLES

```
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
```

FILES

`/usr/bin/spline`

DIAGNOSTICS

When data is not strictly monotone in x, `spline` reproduces the input without interpolating extra points.

SEE ALSO

`graph`(1G), `tplot`(1G).

BUGS

A limit of 1,000 input points is enforced silently.

NAME

split — split a file into pieces

SYNOPSIS

split [-*n*] [*file* [*name*]]

DESCRIPTION

split reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files.

The name of the first output file is *name* with aa appended, and so on lexicographically, up to zz (a maximum of 676 files). *name* cannot be longer than 12 characters. If no output name is given, x is default.

If no input file is given, or if – is given in its stead, then the standard input file is used.

EXAMPLES

```
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

SEE ALSO

bfs(1), csplit(1), fsplit(1).

NAME

ssp — make output single spaced

SYNOPSIS

ssp [-] [*name* ...]

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.

The - option removes all blank lines.

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

SEE ALSO

awk(1), sed(1).

NAME

strings — find the printable strings in an object or other binary file

SYNOPSIS

strings [-] [-o] [-*number*] *file* ...

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. Unless the - flag is given, strings only looks in the initialized data space of object files. If the -o flag is given, then each string is preceded by its offset in the file (in octal). If the -*number* flag is given, then number is used as the minimum string length rather than 4.

strings is useful for identifying random object files and many other things.

EXAMPLES

```
strings obj1
```

will locate the ASCII-character strings in the object file obj1.

FILES

/bin/strings

SEE ALSO

od(1), xstr(1).

BUGS

The algorithm for identifying strings is extremely primitive.

NAME

strip — strip symbol and line number information from an object file

SYNOPSIS

strip [-l] [-r] [-s] [-V] [-x] *file...*

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 following flag options:

- -l Strip line number information only; do not strip any symbol table information.
- -x Do not strip static or external symbol information.
- -r Reset the relocation indexes into the symbol table.
- -s Reset the line number indexes into the symbol table (do not remove). Reset the relocation indexes into the symbol table.
- -V Print the version of the strip command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, strip complains and terminates without stripping *filename* unless the -r flag 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 flag 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.

FILES

/bin/strip
/usr/tmp/str??????

SEE ALSO

as(1), cc(1), ld(1), sdb(1), ar(4), a.out(4).

DIAGNOSTICS

`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` flag was not used; therefore, the symbol table information cannot be stripped.

NAME

stty — set the modes for a terminal

SYNOPSIS

stty [-n *file*] [-a] [-g] [*options*]

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.

With the -n *file* flag option, stty opens 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.)

With the -a flag option, stty reports all of the option settings.

With the -g flag option, stty reports current settings in a form that can be used as an argument to another stty command. Detailed information about the modes listed in the groups "Control Modes", "Input Modes", "Output Modes", "Local Modes", and "Control Assignments" may be found in termio(7). Flag 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.

The options are selected from the following:

Control Modes

parenb (-parenb)	Enable (disable) parity generation and detection.
parodd (-parodd)	Select odd (even) parity.
cs5 cs6 cs7 cs8	Select character size (see termio(7)).
0	Hang up phone line immediately.
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb	Set 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`)	Hang up (do not hang up) modem connection on last close.
`hup` (`-hup`)	Same as `hupcl` (`-hupcl`).
`cstopb` (`-cstopb`)	Use two (one) stop bits per character.
`cread` (`-cread`)	Enable (disable) the receiver.
`clocal` (`-clocal`)	Assume a line without (with) modem control.
`loblk` (`-loblk`)	Block (do not block) output from a non-current layer.

Input Modes

`ignbrk` (`-ignbrk`)	Ignore (do not ignore) break on input.
`brkint` (`-brkint`)	Signal (do not signal) `INTR` on break.
`ignpar` (`-ignpar`)	Ignore (do not ignore) parity errors.
`parmrk` (`-parmrk`)	Mark (do not mark) parity errors (see `termio`(7)).
`inpck` (`-inpck`)	Enable (disable) input parity checking.
`istrip` (`-istrip`)	Strip (do not strip) input characters to seven bits.
`inlcr` (`-inlcr`)	Map (do not map) NL to CR on input.
`igncr` (`-igncr`)	Ignore (do not ignore) CR on input.
`icrnl` (`-icrnl`)	Map (do not map) CR to NL on input.
`iuclc` (`-iuclc`)	Map (do not map) uppercase alphabetics to lowercase on input.
`ixon` (`-ixon`)	Enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.
`ixany` (`-ixany`)	Allow any character (only DC1) to restart output.
`ixoff` (`-ixoff`)	Request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

`opost` (`-opost`)	Post-process output (do not post-process output; ignore all other output modes).
`olcuc` (`-olcuc`)	Map (do not map) lowercase alphabetics to uppercase on output.
`onlcr` (`-onlcr`)	Map (do not map) NL to CR-NL on output.
`ocrnl` (`-ocrnl`)	Map (do not map) CR to NL on output.
`onocr` (`-onocr`)	Do not (do) output a CR at column zero.
`onlret` (`-onlret`)	On the terminal, NL performs (does not perform) the CR function.
`ofill` (`-ofill`)	Use fill characters (use timing) for delays.
`ofdel` (`-ofdel`)	Fill characters are DELs (NULs).
`cr0 cr1 cr2 cr3`	Select style of delay for returns (see `termio`(7)).
`nl0 nl1`	Select style of delay for linefeeds (see `termio`(7)).
`tab0 tab1 tab2 tab3`	Select style of delay for horizontal tabs (see `termio`(7)).
`bs0 bs1`	Select style of delay for backspaces (see `termio`(7)).
`ff0 ff1`	Select style of delay for form-feeds (see `termio`(7)).
`vt0 vt1`	Select style of delay for vertical tabs (see `termio`(7)).

Local Modes

`isig` (`-isig`)	Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.
`icanon` (`-icanon`)	Enable (disable) canonical input (ERASE and KILL processing).
`xcase` (`-xcase`)	Canonical (unprocessed) upper/lowercase presentation.

`echo` (`-echo`) Echo back (do not echo back) every character typed.

`echoe` (`-echoe`) Echo (do not echo) ERASE character as a backspace-space-backspace string.

Note: This mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

`echok` (`-echok`) Echo (do not echo) NL after KILL character.

`lfkc` (`-lfkc`) Same as `echok` (`-echok`); obsolete.

`echonl` (`-echonl`) Echo (do not echo) NL.

`noflsh` (`-noflsh`) Disable (enable) flush after INTR, QUIT, or SWTCH.

`stwrap` (`-stwrap`) Disable (enable) truncation of lines longer than 79 characters on a synchronous line.

`stflush` (`-stflush`) Enable (disable) flush on a synchronous line after every `write`(3).

`stappl` (`-stappl`) Use application mode (use line mode) on a synchronous line.

Control Assignments

`erase` *c* Set erase character to *c* (by default, set to DELETE in the A/UX standard distribution).

`kill` *c* Set kill character to *c* (by default, set to CONTROL-U in the A/UX standard distribution).

`intr` *c* Set interrupt character to *c* (by default, set to CONTROL-C in the A/UX standard distribution).

`quit` *c* Set quit character to *c* (by default, set to CONTROL-\ in the A/UX standard distri-

bution).

`swtch` *c* — Set switch character to *c* (by default, set to CONTROL-\ in the A/UX standard distribution).

`eof` *c* — Set EOF character to *c* (by default, set to CONTROL-D in the A/UX standard distribution).

`eol` *c* — Set EOL character to *c* (by default, set to CONTROL-\ in the A/UX standard distribution).

`min` *c* — Set min character to *c* (`min` is used only with `-icanon`; (see `termio`(7)).

`time` *c* — Set 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* — Set line discipline to *i* ($0 \leq i < 127$).

BSD 4.2 Compatible Features

`susp` *c* — Set the suspend character to *c*. When typed, the suspend character sends `SIGTSTP` to the current process group.

`dsusp` *c* — When set and a program attempts to read terminal input, `SIGTSTP` is sent to the current process group.

`tostop` (`-tostop`) — When set, background processes which write on the control tty will be stopped until brought into foreground by the shell.

Combination Modes

`evenp` or `parity` — Enable `parenb` and `cs7`.

`oddp` — Enable `parenb`, `cs7`, and `parodd`.

`-parity`, `-evenp`, or `-oddp`
: Disable `parenb`, and set `cs8`.

`raw` (`-raw` or `cooked`)
: Enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).

`nl` (`-nl`)
: Unset (set) `icrnl` and `onlcr`. In addition `-nl` unsets `inlcr`, `igncr`, `ocrnl`, and `onlret`.

`lcase` (`-lcase`)
: Set (unset) `xcase`, `iuclc`, and `olcuc`.

`LCASE` (`-LCASE`)
: Same as `lcase` (`-lcase`).

`tabs` (`-tabs` or `tab3`)
: Preserve (expand to spaces) tabs when printing.

`ek`
: Reset ERASE and KILL characters back to normal DELETE and CONTROL-U.

`sane`
: Reset all modes to some reasonable values.

term
: Set all modes suitable for the terminal type *term*, where *term* is one of `tty33`, `tty37`, `vt05`, `tn300`, `ti700`, or `tek`.

Hardware-specific Modes

`modem` (`-modem`)
: Enable (disable) 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)` Enable (disable) "European style" modem control. Similar to `modem`, as described previously. Refer to `termio(7)` for further information.

`dtrflow (-dtrflow)`
`hxctl (-hxctl)` Enable 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)` Enable 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`

SEE ALSO

`tabs(1)`, `ioctl(2)`, `termio(7)`.

NAME

style — analyze surface characteristics of a document

SYNOPSIS

style [-ml] [-mm] [-a] [-e] [-l *num*] [-r *num*] [-p] [-P] *file...*

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. The default macro package -ms may be overridden with the flag -mm. The flag -ml, which causes deroff to skip lists, should be used if the document contains many lists of nonsentences. The other flag options are used to locate sentences with certain characteristics.

-a print all sentences with their length and readability index.

-e print all sentences that begin with an expletive.

-p print all sentences that contain a passive verb.

-l *num*
print all sentences longer than *num*.

-r *num*
print all sentences whose readability index is greater than *num*.

-P print parts of speech of the words in the document.

FILES

/usr/ucb/style
/usr/lib/style1
/usr/lib/style2
/usr/lib/style3

SEE ALSO

deroff(1), diction(1), spell(1).

BUGS

Use of nonstandard formatting macros may cause incorrect sentence breaks.

NAME

su — substitute user ID

SYNOPSIS

su [-] [*name*[*arg* ...]]

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.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like sh(1), an *arg* of the form -c *string* executes *string* via the shell and an argument of -r will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like sh(1). If the first argument to su is a -, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is -, thus causing 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 trip
```

would cause the system to prompt for `trip`'s password; if the password is typed in correctly, `trip`'s identity is substituted.

To become user `bin` while retaining the previously exported environment, execute

```
su bin
```

To become user `bin` but change the environment to what would be expected if `bin` had originally logged in, execute

```
su - bin
```

To execute *command* with the temporary environment and permission of user `bin`, type

```
su - bin -c "command args"
```

FILES

```
/bin/su
/etc/passwd
/etc/profile
$HOME.profile
/usr/adm/sulog
```

SEE ALSO

csh(1), env(1), ksh(1), login(1), sh(1), passwd(4), profile(4), environ(5).

NAME

subj — generate a list of subjects from a document

SYNOPSIS

subj *file* ...

DESCRIPTION

subj searches *files* 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.

FILES

/usr/bin/subj

SEE ALSO

mm(1), ndx(1), troff(1).

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.

BUGS

subj 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.

NAME

sum — calculate a checksum

SYNOPSIS

sum [-r] *file...*

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. The flag option -r causes an alternate algorithm to be used in computing the checksum.

EXAMPLES

```
sum filea
```

produces the checksum and the block count of `filea`.

FILES

/bin/sum

SEE ALSO

sumdir(1), wc(1).

DIAGNOSTICS

`Read error` is indistinguishable from end-of-file on most devices; check the block count.

NAME

sumdir — sum and count characters in the files in the given directories

SYNOPSIS

sumdir [*directories*]

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

```
sumdir man1
```

produces the checksum and the character count of the files in the directory man1.

FILES

/usr/bin/sumdir

SEE ALSO

sum(1).

NAME

sync — update 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

```
sync
```

should be typed to flush all internal disk buffers, before bringing down the system.

FILES

`/bin/sync`

SEE ALSO

`shutdown`(1M), `sync`(2).

NAME

sysline — display system status on status line of a terminal

SYNOPSIS

sysline [-b] [-c] [-d] [-e] [-h] [-D] [-i] [-l] [-m] [-p] [-q] [-r] [-s] [-j] [-H *remote*] [+*N*]

DESCRIPTION

sysline runs in the background and periodically displays system status information on the status line of the terminal. Not all terminals contain a status line. Those that do include the h19, c108, aaa, vt100, tvi925/tvi950 and Freedom 100.

Note: The Macintosh II does not have a status line.

If no flags 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 process (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 is 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.

The following flag options may be used on the command line:

-b	Beep once every half hour and twice every hour, just like those obnoxious watches you keep hearing.
-c	Clear the status line for 5 seconds before each redisplay.
-d	Debug mode – print status line data in human-readable format
-D	Print out the current day/date before the time.
-e	Print out only the information. Do not print out the control commands necessary to put the information

on the bottom line. This flag option is useful for putting the output of `sysline` onto the mode line of an `emacs` window.

`-H` *remote* Print 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.

`-h` Print out the host machine's name after the time.

`-l` Don't print the names of people who log in and out.

`-m` Don't check for mail.

`-p` Don't report the number of process which are runnable and suspended.

`-r` Don't display in reverse video.

`+N` Update the status line every N seconds. The default is 60 seconds.

`-q` Don't print out diagnostic messages if something goes wrong when starting up.

`-i` Print 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.

`-s` Print "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 flag option will disable the adding of the blanks.

`-j` Force the sysline output to be left-justified even on terminals capable of cursor movement on the status line.

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 dis-

able 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.

FILES

/usr/ucb/sysline	
/etc/utmp	names of people who are logged in
/dev/kmem	contains process table
/usr/spool/rwho/whod.*	who/uptime information for remote hosts
${HOME}/.who	information to print on bottom line
${HOME}/.syslinelock	when it exists, sysline will not print

SEE ALSO

ps(1), pstat(1).

BUGS

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.

NAME

`systemfolder` — create a personal System Folder

SYNOPSIS

`systemfolder [-f]`

DESCRIPTION

`systemfolder` allows a user to create a personal Macintosh® System Folder in his or her home directory. If a personal System Folder already exists, it is updated with any files in the global System Folder that are not present in the personal Folder.

FLAG OPTIONS

`systemfolder` interprets one flag option, the force option.

`-f` By default, the System file is not updated if it already exists in the personal System Folder. If this option is used, this forces the update of the System file.

FILES

```
$HOME/System Folder
/mac/sys/System Folder/*
```

NAME

tabs — set tabs on a terminal

SYNOPSIS

tabs [*tabspec*] [+m[*n*]] [–T*type*]

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, 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 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.

tabspec may be any 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 *--file* flag 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 *--file* flag 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` flag option for high-speed output. Another special case is the value `-0`, implying no tabs at all.

n1,n2,...
The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists `1,10,20,30` and `1,10,+10,+10` are considered identical.

--file If the name of a file is given, `tabs` reads the first line of the file, searching for a format specification. If it finds

one there, it sets the tab stops according to it, otherwise it sets them as -8. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the pr(1) command:

tabs --*file*; pr *file*

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

-T*type* tabs usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in term(5). If no -T flag is supplied, tabs 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.

+m*[n]* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If +m is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (left-most) margin on most terminals is obtained by +m0. The margin for most terminals is reset only when the +m flag is given explicitly.

Tab and margin setting is performed via the standard output.

EXAMPLES

```
tabs -c
```

will send commands to the terminal to set the tabs for COBOL format remotely.

```
tabs 6,12,18
```

will set tabs in columns 6, 12 and 18.

```
tabs -10
```

will set tabs in columns 11, 21, 31, 41, 51, 61, and 71.

DIAGNOSTICS

illegal tabs	when arbitrary tabs are ordered incorrectly.
illegal increment	when a zero or missing increment is found in an arbitrary specification.

`unknown tab code`	when a predefined code cannot be found, where predefined codes include: `-a -a2 -c -c2 -c3 -f -p -s -u`
`can't open`	if *--file* option is used, and file can't be opened.
`file indirection`	if *--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`

SEE ALSO

`nroff`(1), `pr`(1), `tset`(1), `term`(4), `environ`(5).

BUGS

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.

`tabs` clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

NAME

tail — deliver the last part of a file

SYNOPSIS

tail [±[*number*][lbc[f]]] [*file*]

DESCRIPTION

tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *–number* from the end of the input (if *number* is null, the value 10 is assumed). *number* is counted in units of lines, blocks, or characters, according to the appended option l, b, or c. When no units are specified, counting is by lines.

With the –f ("follow") flag option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. You must interrupt tail to escape this loop.

EXAMPLES

```
tail -f jack
```

will 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.

```
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.

FILES

/bin/tail

SEE ALSO

cat(1), dd(1), head(1), more(1), pg(1).

BUGS

tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

NAME

talk — talk to another user

SYNOPSIS

talk *person* [*ttyname*]

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 *person* is just the person's login name. If you wish to talk to a user on another host connected via Ethernet to a local network running B-NET software, then *person* is of the form

host!*user* or
host.*user* or
host:*user* or
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.

When 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 to whom you wish to talk. 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. Typing CONTROL-L will cause the screen to be reprinted, while your erase and kill characters will work in talk as normal. To exit, 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.

FILES

`/usr/bin/talk`	
`/etc/hosts`	to find the recipient's machine
`/etc/utmp`	to find the recipient's tty

SEE ALSO

`mail`(1), `mesg`(1), `who`(1), `write`(1).
"Using B-NET" in *A/UX Communications User's Guide*.

NAME

`tar` — copy files to or from a `tar` archive

SYNOPSIS

`tar` [*key*] [*file...*]

DESCRIPTION

`tar` saves and restores files within an archive, which frequently is magnetic tape media or floppy disks. Its actions are controlled by the *key* argument. The *key* is a string of characters containing a maximum of one function letter and possibly one or more function modifiers.

Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory. `tar` does not follow symbolic links.

The function portion of the key is specified by one of the following letters:

`r` The named *files* are written on the end of the tape. This may not work on all media. It requires the ability to "seek". The `c` function implies this function.

`x` The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. If a named file on tape does not exist on the system, the file is created with the same mode as the one on tape, except that the set-user-ID and set-group-ID bits are not set unless you are superuser. If the files exist, their modes are not changed except for the bits described above. The owner, group, and modification time are restored (if possible). If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.

`t` The names of all the files on the tape are listed.

`u` The named *files* are added to the tape if they are not already there or have been modified since last written on that tape.

`c` Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This command implies the `r` function.

The following characters may be used in addition to the letter that selects the desired function.

ns Where *n* is a tape drive number (`0,...,7`), and *s* is the density (`l` - low (800 bpi), `m` - medium (1600 bpi), or `h` - high (6250 bpi)). This modifier selects the drive on which the tape is mounted. The default is `0m`.

`i` Causes `tar` to ignore symbolic links.

`v` Normally, `tar` does its work silently. The `v` (verbose) flag option causes it to type the name of each file it treats, preceded by the function letter. With the `t` function, `v` gives, in addition to the name, information about the tape entries.

`w` Causes `tar` to print the action to be taken, followed by the name of the file, and then to wait for the user's confirmation. If a word beginning with `y` is given, the action is performed. Any other input means "no".

`f` Causes `tar` to use the next argument as the name of the archive instead of `/dev/mt/0m`. If the name of the file is `-`, `tar` writes to the standard output or reads from the standard input, whichever is appropriate. Thus, `tar` may be used as the head or tail of a pipeline. `tar` may also be used to move hierarchies with the command:

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

`b` Causes `tar` to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This flag option should only be used with raw magnetic tape archives (see `f`). The block size is determined automatically when reading tapes (key letters `x` and `t`).

`l` Tells `tar` to complain if it cannot resolve all of the links to the files being dumped. If `l` is not specified, no error messages are printed.

`m` Tells `tar` not to restore the modification times. The modification time of the file will be the time of extraction.

o Causes extracted files to take on the user and group identifier of the user running the program rather than those on the tape.

If more than one flag is used that requires an argument, the arguments must be supplied in the same order that the flags were specified.

This version of `tar` is capable of writing more than one tape or disk. The user will be prompted to change media when necessary. The next two flag options are used for tapes; the last is for disks.

d Causes `tar` to use the next argument as the tape's density. The default density is 1600 bpi.

s Causes `tar` to use the next argument as the tape's length in feet. The default length is 2300 feet.

B Causes `tar` to use the next argument as the number of 512-byte blocks on the disk.

EXAMPLES

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

will copy directories from one directory tree (`fromdir`) to another (`todir`).

FILES

`/usr/bin/tar`	
`/dev/rmt/*`	
`/dev/mt/*`	
`/tmp/tar*`	
`/bin/mkdir`	build directories during recovery
`/bin/pwd`	get working directory name

SEE ALSO

`ar`(1), `cpio`(1), `dd`(1), `tp`(1), `dump.bsd`(1M), `tar`(4).

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints that enough memory is not available to hold the link tables.

BUGS

There is no way to ask for the *n*th occurrence of a file.
Tape errors are handled ungracefully.
The `u` flag option can be slow.
The `b` flag 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` flag 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 archive special devices.

NAME

tbl — format tables for nroff or troff

SYNOPSIS

tbl [-TX] [*file*...]

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).

.TS is followed by global options. The available global options are:

center	center the table (default is left adjust);
expand	make the table as wide as the current line length;
box	enclose the table in a box;
doublebox	enclose the table in a double box;
allbox	enclose each item of the table in a box;
tab(*x*)	use 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 center item within the column;
- r right adjust item within the column;
- l left adjust item within the column;
- n numerically adjust item in the column: units positions of numbers are aligned vertically;
- s span previous item on the left into this column;
- a center longest line in this column and then left adjust all other lines in this column with respect to that centered line;

^ span down previous entry in this column;
_ replace this entry with a horizontal line;
= replace 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.

The -TX flag option 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).

If no file names 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.

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^I Households
^I Number^I Size
=
Bedminster^I 789^I 3.26
Bernards Twp.^I 3087^I 3.74
Bernardsville^I 2018^I 3.30
Bound Brook^I 3425^I 3.04
```

```
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

FILES

`/bin/tbl`

SEE ALSO

eqn(1), mm(1), mvt(1), nroff(1), troff(1), mm(5), ms(5), mv(5).

"tbl Reference" in *AIUX Text Processing Tools.*

BUGS

See BUGS under nroff(1).

NAME

tc — interpret troff output for use at a vintage display device

SYNOPSIS

tc [-t] [-o *list*] [-a *n*] [-e] [*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*	Send *cmd* to the shell.
e	Invert the state of the screen erase.
n	Print page *n* (previously printed).
–*n*	Skip backward *n* pages.
o*list*	Set -o list to *list*.
p	Print current page again.
a*n*	Set the aspect ratio to *n*.
?	Print list of available options.

The flag options are

-t	Do not wait between pages (for directing output into a file).
-o *list*	Print 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*.
-a *n*	Set the aspect ratio to *n*; default is 1.5.

-e Do not erase before each page.

FILES

/usr/bin/tc

SEE ALSO

4014(1), nroff(1), tplot(1G), troff(1).

BUGS

Font distinctions are lost.

tc needs a -w flag option to wait for input to arrive.

NAME

tcb — block data to 8K for tc output

SYNOPSIS

command-line | tcb >/dev/rmt/tc*x*

DESCRIPTION

tcb reads standard input and writes standard output in a blocking format suitable for the Apple SC 40 Tape Backup. 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 data tcb will be lost.

The following example illustrates how to create a tar archive on cartridge tape. Substituting the SCSI ID of your tape cartridge drive for *x*, enter

```
tar cvf - . | tcb >/dev/rmt/tcx
```

To create an archive that will be larger than one tape, use the *size* (B) flag option for tar. This flag option allows you to specify the capacity of the tape cartridge in terms of 512-byte blocks. This may be computed 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 the use of a larger input buffer size, as long as it is a multiple of 8K:

```
dd if=/dev/rmt/tcx ibs=20x8K |tar xvf -
```

SEE ALSO

dd(1), tar(1), tc(7).

NAME

tee — pipe fitting

SYNOPSIS

tee [-i] [-a] [*file*] ...

DESCRIPTION

tee transcribes the standard input to the standard output and makes copies in the *files*. The -i flag option ignores interrupts; the -a flag option causes the output to be appended to the *files* rather than overwriting them.

EXAMPLES

```
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

NAME

telnet — user interface to the TELNET protocol

SYNOPSIS

telnet [*host* [*port*]]

DESCRIPTION

telnet is used to communicate with another host using the TELNET protocol. 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. If it is invoked with arguments, it performs an open command (see below) with those arguments.

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*]
Open a connection to the named *host*. If no *port* number is specified, `telnet` will attempt 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`
Close a TELNET session and return to command mode.

`quit`
Close any open TELNET session and exit `telnet`. An end of file (in command mode) will also close a session and exit.

`z` Suspend `telnet`. This command only works when the user is using the `csh`(1).

`mode` *type*
type is 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`
Show 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*]
Get help. With no arguments, `telnet` prints a help summary. If a command is specified, `telnet` will print 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 ini-

tial value of the EOF character is taken to be the terminal's EOF character.

`toggle` *arguments...*

Toggle (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.

FILES

`/usr/bin/telnet`

SEE ALSO

`cu`(1), `ftp`(1N), `rlogin`(1N), `tip`(1C), `uucp`(1C), `telnetd`(1M).

BUGS

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.

NAME

`test` — condition evaluation command

SYNOPSIS

`test` [*expr*]

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. `test` 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

`test` is typically used in shell scripts (`sh`(1)), as in the following example, which prints the message "`foo is a directory`" if it is found to be one when `test` is run.

```
if test -d foo
then
        echo "foo is a directory"
fi
```

SEE ALSO

`find`(1), `ksh`(1), `sh`(1).
"Bourne Shell Reference" and "Korn Shell Reference" in *A/UX User Interface*.

NAME

TextEditor — mouse-based text editor

SYNOPSIS

TextEditor [*filename*]

DESCRIPTION

TextEditor is a mouse-based editor for use with both Macintosh® and A/UX® files of type TEXT. It runs 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 instead of with keyboard commands.

You can invoke 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 entering launch TextEditor in the A/UX command line. For 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 the text in the active window by using the scroll bar that runs along its right side, as described in *A/UX Essentials* .

Files created or touched by TextEditor are saved as text-only files of type TEXT. They may contain tab and newline characters but no other formatting information. This file structure is compatible with 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 %*filename* 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 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. Move the mouse so that you 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 tab and newline, 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 under the Apple® menu) and use 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, instead of 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. Just choose Undo from the Edit menu immediately after the faulty operation.

The next section "Text Selection" tells you how to select the text to be edited; "Menu Commands" lists the operations you can perform.

TEXT SELECTION

There are several ways you can select a section of existing text for a TextEditor editing operation.

Double-clicking

When you position the pointer on a word and press the mouse button twice in rapid succession, TextEditor selects that word. This 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 under-

score 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 below, it 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 is called "triple-clicking."

Dragging
When you move the pointer over text from one place to another while pressing and holding down the mouse button, TextEditor selects all the text the pointer passes over until you release the mouse button. This 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 showing, TextEditor automatically scrolls the window to show more text.

Shift Selection
When you move the pointer to a place other than the current insertion point and then click while holding down SHIFT, 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 showing.

Marker Selection
"Mark Menu" in the later section, "Menu Commands" describes how you can create names for selections of text. To select a piece of text you have previously named, you just choose its name from the Mark menu.

Enclosed Text Selection
When you double-click one of the following pairs of polarized enclosing characters, TextEditor selects all text between it and the matching character. These characters are:

```
( )     [ ]     { }
```

This method of selecting text works both backward and forward. For example, if you click 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 one of the following pairs of nonpolarized enclosing characters, `TextEditor` selects all text between it and the next occurrence of the same character forward in the text. These characters are:

" " ' ' ` ` / / \ \

When making a selection from both polarized and nonpolarized enclosing characters, `TextEditor` ignores all characters except the correct match and searches to the beginning or end of the document. The resulting selection does not include the enclosing characters themselves. If `TextEditor` does not find a match, it selects only the character originally 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. 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 (frontmost or topmost) window.

Many menu actions can be invoked from the keyboard by holding down COMMAND (not CONTROL) and typing a character. The character required is shown beside the Command-key symbol in the menu display. Such Command-key equivalents may be entered as lowercase; you don't need to hold down SHIFT as well.

The following sections 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... Create 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 Drive button in this dialog box, TextEditor searches for a different hard disk or floppy disk. When you click the New button, it creates the file and opens it in a new, active TextEditor window. The Command-key equivalent for the New menu item is COMMAND-N.

The New... dialog also contains radio buttons that let you specify whether the resulting file will be saved with its formatting information or as text only.

Open... Open an existing text file from disk. This menu action first displays a hierarchical list of all files of type TEXT that are available in your immediate system. 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 box. 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 Close the active window and remove it from view. You can recall the window later by using the Window menu. This menu action does not save the window contents to disk. The Command-key equivalent for the Close menu item is COMMAND-W.

If you have not previously saved the file, this menu action displays a dialog that lets you specify whether the file being closed will be saved, and if it is saved whether it is with formatting information or as text only.

Save	Save the contents of the active window to disk in the file that was originally opened, without closing the window. This menu item is grayed 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...	Save the contents of the active window to disk into a different file than the one originally opened, and start to edit the new file. The original file is closed with its original name but without any new changes being saved to it. This menu action displays a dialog box that lets you enter the new filename and specify its directory location, as well as radio buttons that let you specify whether or not formatting information is saved. The active window then shows the name and contents of the new file and subsequent Save actions save the contents of the window to it.
Save a Copy...	Act the same as Save As, but continue editing the original file under the original name.
Revert to Saved	Discard all changes to the contents of the active window since it was last saved. This menu item is grayed if the contents of the file have not been changed since the last Save action.
Page Setup	Display a dialog box that lets you set the paper size, orientation, and reduction or enlargement for subsequent printing actions.
Print Window	Print text from the active window. If part of the text is currently selected, `TextEditor` prints only the selection; otherwise, it prints the entire document 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, just described, to specify paper size, orientation, and scale.
Quit	Exit `TextEditor` and return 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 items in the Edit menu help you move text around and perform certain global formatting actions.

Undo Reverse 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. The Command-key equivalent for the Undo menu item is COMMAND-Z.

Cut Copy the currently selected text in the active window to the Clipboard and then delete it from the window. The Command-key equivalent for the Cut menu item is COMMAND-X.

Copy Copy 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 Replace the currently selected text in the active window with the contents of the Clipboard. If there is no current selection, Paste inserts the contents of the clipboard at the current insertion point. The Command-key equivalent for the Paste menu item is COMMAND-V.

Clear Delete the currently selected text from the active window. The key equivalent for the Clear menu item is DELETE.

Select All Select the entire document that is in the active window. The Command-key equivalent for the Select All menu item is COMMAND-A.

Show Clipboard Display a new, active `TextEditor` window that displays the contents of the Clipboard, if any.

Format... Display a dialog box that lets you set typography and indentation for the entire document that is in the active window.

Scrolling list (fields) in the Format dialog box let you select a type font and size for the active window by clicking items in the lists.

The Auto Indent check box in the Format dialog box toggles auto-indenting on and off, with the X-mark indicating it is on. When auto-indenting is on, pressing RETURN aligns text to the left margin of the previous 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 check box in the Format dialog box toggles invisible character display on and off, with the X-mark indicating it is on. When it 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 (rotated L's), 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 — Align the left margin of all the currently selected text in the active window to the top line of the selection.

Shift Left — Move the currently selected text in the active window one tab distance 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 SHIFT, the movement becomes one space instead of one tab.

Shift Right — Perform the same action as Shift Left, but move 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 an option so the check box is checked:

Literal	Find the exact string you entered, wherever it may appear, even if it is part of another string.
Entire Word	Find the string you entered only if it constitutes an entire word. The determination of word boundaries is the same as with double-clicking, described earlier in "Text Selection." The Entire Word and Literal options are mutually exclusive.
Case Sensitive	Find the string you entered only if the uppercase and lowercase status of all letters in the found string is the same.
Search Backward	Search 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 SHIFT when you start a search operation.
Wrap-around Search	Search forward to the end of the document, then start searching again from the beginning to the current selection or insertion point. If Search Backwards is also selected, Wrap-around Search does the same in the reverse direction.

The default values for searching are Literal on, Entire Word off, Case Sensitive off, Search Backwards off, and Wrap-around Search off. Whenever a search fails, `TextEditor` tells you by sounding a beep.

The Find menu contains the following menu items:

Find...	Find the next occurrence of the string you specify in the text box. `TextEditor` scrolls the active window to that part of the

document and selects the text it has found. The Command-key equivalent for the Find menu item is COMMAND-F.

Find Same — Repeat the most recent Find operation. The Command-key equivalent for the Find Same menu item is COMMAND-G.

Find Selection — Find the next occurrence of the currently selected text. The Command-key equivalent for the Find Selection menu item is COMMAND-H.

Display Selection — Scroll the active window to show the currently selected text.

Replace... — Find the next occurrence of the string you specify in a text box and replace it with another string that you also specify. The Command-key equivalent for the Replace menu item is COMMAND-R.

Replace Same — Repeat 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 find them easily later. They also make it easy to select large pieces of text, as explained earlier in "Text Selection."

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 you have created for the currently active window. The Mark and Unmark menu items do the following:

Mark... — Display 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 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... Display a dialog box that lets you remove unwanted markers. The Unmark dialog box shows you a scrolling 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 help you arrange and recall 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 full pathnames of all windows currently displayed in TextEditor.

The Tile Windows and Stack Windows menu items, in the top part of the Window menu, do the following:

Tile Windows Arrange the currently open windows vertically, so that at least part of the contents of each one is visible.

Stack Windows Arrange the currently open windows in a diagonally staggered overlapping pattern, with the currently active window in front. The active window is the only one whose contents are visible.

When you choose one of the items in the lower window list, TextEditor makes it the active window. The names of currently displayed windows are listed in the order they were originally displayed. In addition, they are marked as follows:

Check mark The currently active window

Round bullet The window that was active just before the currently active window, and hence is second to the front

Underline Any window containing changes that have not yet been saved

FILES

/usr/toolboxbin/TextEditor
/usr/toolboxbin/%TextEditor

SEE ALSO

vi(1), ed(1).

See *MPW 3.0 Reference* for a description of a similarly constructed mouse-based text editor.

NAME

tftp — trivial file transfer program

SYNOPSIS

tftp [*host*]

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. The remote *host* may be specified on the command line, in which case tftp uses *host* as the default host for future transfers (see the connect command below).

COMMANDS

Once tftp is running, it issues the prompt tftp> and recognizes the following commands:

connect *host-name* [*port*]
: Set the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections betweeen 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*
: Set the mode for transfers; *transfer-mode* may be one of ascii or binary. The default is ascii.

put *file*
put *localfile remotefile*
put *file1 file2 ... fileN remote-directory*
: Put 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 *file1 file2 ... fileN*
: Get 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`
Exit `tftp`. An end of file also exits.

`verbose`
Toggle verbose mode.

`trace`
Toggle packet tracing.

`status`
Show current status.

`rexmt` *retransmission-timeout*
Set the per-packet retransmission timeout, in seconds.

`timeout` *total-transmission-timeout*
Set the total transmission timeout, in seconds.

`ascii`
Shorthand for "mode ascii"

`binary`
Shorthand for "mode binary"

`?` [*command-name* ...]
Print help information.

FILES

`/usr/bin/tftp`

BUGS

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.

NAME

time — time a command

SYNOPSIS

time *command*

DESCRIPTION

time executes the *command*; afterward, time prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on the standard error output.

EXAMPLES

```
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

SEE ALSO

csh(1), timex(1), times(2).

NAME

timex — time a command; report process data and system activity

SYNOPSIS

timex [-o] [-p[fhkmrt]] [-s] *command*

DESCRIPTION

timex sends the given *command* to the shell for execution; timex 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.

FLAG OPTIONS

The flag options interpreted by timex are:

-p List process accounting records for *command* and all its children. 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.

-o Report the total number of blocks read or written and total characters transferred by *command* and all its children.

-s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. timex reports all the data items listed in sar(1).

EXAMPLES

```
timex ps -el
```

runs the ps command (with the correct flag options), then produces statistics concerning the command and system activity during the command to the standard error output.

FILES

/usr/bin/timex
/usr/lib/sa/timex

SEE ALSO

acctcom(1M), sar(1), time(1).

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.

NAME

tip — connect to a remote system

SYNOPSIS

tip [-v] [-*speed*] *system-name*

tip [-v] [-*speed*] *phone-number*

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.

~	Drop 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*]	Change directory to *name* (no argument implies change to your home directory).
~!	Escape to a shell (exiting the shell will return you to tip).
~>	Copy file from local to remote. tip prompts for the name of a local file to transmit.
~<	Copy 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*]	Send 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*]	Take 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.

~| Pipe the output from a remote command to a local UNIX process. tip 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.

~# Send 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 Set a variable (see the discussion later in this section).

~CONTROL-Z Stop tip (available only with job control).

~? Get a summary of the tilde escapes

tip 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. If the baud rate is specified as 300 baud but no system name is supplied, then tip assumes that a host with the name tip300 exists in the /etc/remote file. Similarly, if no speed is specified but a telephone number is provided, then tip looks for a host with the name tip0. Refer to remote(4) for a full description.

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.

tip 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.

`tip` 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). The `-v` flag option causes `tip` to display the sets as they are made. 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. `tip` 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. 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.

FILES

`/usr/ucb/tip`	
`/etc/dialup`	modem escape sequences
`/etc/remote`	global system descriptions
`/etc/phones`	global telephone number data base
`${REMOTE}`	private system descriptions
`${PHONES}`	private telephone numbers
`~/.tiprc`	initialization file
`/usr/spool/uucp/LCK*`	lock file to avoid conflicts with `uucp`

SEE ALSO

`cu`(1C), `ftp`(1N), `telnet`(1N), `uucp`(1C), `dialup`(4), `remote`(4), `phones`(4).
"Using `cu`" in *A/UX Communications User's Guide*.

BUGS

The full set of variables is undocumented and probably should be pared down.

NAME

touch — update access and modification times of a file

SYNOPSIS

touch [–a] [–c] [–m] [*mmddhhmm* [*yy*]] *file* ...

DESCRIPTION

touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see date(1)) the current time is used. The –a and –m flag options cause touch to update only the access or modification times respectively (default is –am). The –c flag option silently prevents touch from creating the file if it did not previously exist.

The return code from touch is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

EXAMPLES

```
touch filea fileb
```

sets the "date last modified" of the two files to the current date.

FILES

/bin/touch

SEE ALSO

date(1), utime(2).

BUGS

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.

NAME

tp — copy files to or from a tp archive

SYNOPSIS

tp [*key*] [*name ...*]

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 key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

tp is useful for importing tapes made on older systems.

The function portion of the key is specified by one of the following letters:

r — The named files are written on 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 no file argument is given, . is the default.

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.

d — Deletes the named files from the tape. At least one name argument must be given. This function is not permitted on magnetic tapes.

x — Extracts the named files from the tape to the file system. The owner and mode are restored. If no file argument is given, the entire contents of the tape are extracted.

t — Lists the names of the specified files. If no file argument is given, the entire contents of the tape is listed.

The following characters may be used in addition to the letter which selects the function desired.

`m` Specifies magnetic tape as opposed to DECtape.

`0,...,7`
This modifier selects the drive on which the tape is mounted. For DECtape, `x` is default; for magnetic tape `0` is the default.

`v` Normally `tp` does its work silently. The `v` (verbose) flag option causes it to type the name of each file it treats preceded by the function letter. With the `t` function, `v` gives more information about the tape entries than just the name.

`c` Means a fresh dump is being created; the tape directory is cleared before beginning. Usable only with `r` and `u`. This flag option is assumed with magnetic tape since it is impossible to selectively overwrite magnetic tape.

`i` Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

`f` Use the first named file, rather than a tape, as the archive. This flag option is known to work only with `x`.

`w` Causes `tp` to pause before treating each file, type the indicative letter and the file name (as with `v`) and await 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. In the `x` function, files previously asked about have been extracted already. With `r`, `u`, and `d`, no change has been made to the tape.

EXAMPLES

```
tp x file1
```

extracts `file1` from a `tp` formatted magnetic tape mounted on drive 0.

FILES

```
/bin/tp
/dev/tap?
/dev/mt?
```

SEE ALSO

ar(1), cpio(1), tar(1), dump.bsd(1M).

DIAGNOSTICS

Several; the nonobvious one is `Phase error`, which means the file changed after it was selected for dumping but before it was dumped.

BUGS

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(1) avoids the problem.

`tp` does not copy zero-length files to tape.

NAME

tplot — interpret plotter instructions for use at a vintage display device

SYNOPSIS

tplot [-T*terminal* [-e *raster*]]

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*. If no *terminal* is specified, the environment parameter $TERM (see environ(5)) is used. Known *terminals* are:

300	DASI 300.
300S	DASI 300s.
450	DASI 450.
4014	Tektronix 4014.
ver	Versatec D1200A. This version of plot places a scan-converted image in /usr/tmp/raster$$ and sends the result directly to the plotter device, rather than to the standard output. The -e flag option causes a previously scan-converted file *raster* to be sent to the plotter.

EXAMPLES

```
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
/usr/lib/t300
/usr/lib/t300s
/usr/lib/t450
/usr/lib/t4014
/usr/lib/vplot
/usr/tmp/raster$$

SEE ALSO

plot(3X), plot(4), term(4).

NAME

tput — query terminfo database

SYNOPSIS

tput [-T*type*] *capname*

DESCRIPTION

tput uses the terminfo(4) database to make terminal-dependent capabilities and information available to the shell. tput generates a string if the attribute I(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.

-T*type* indicates the type of terminal. Normally this flag is unnecessary, as the default is taken from the environment variable $TERM.

capname indicates the attribute from the terminfo database. See terminfo(4).

EXAMPLES

```
tput clear
```

Echo clear-screen sequence for the current terminal.

```
tput cols
```

Print the number of columns for the current terminal.

```
tput -T450 cols
```

Print the number of columns for the 450 terminal.

```
bold='tput smso'
```

Set shell variable bold to standout mode sequence for current terminal. This might be followed by a prompt:

```
echo "${bold}Please type in your name: \c"
```

```
tput hc
```

Set exit code to indicate if current terminal is a hardcopy terminal.

FILES

/usr/bin/tput	
/usr/lib/terminfo/?/*	Terminal descriptor files
/usr/include/term.h	Definition files

/usr/include/curses.h

DIAGNOSTICS

tput prints error messages and returns the following error codes on error:

-1	Usage error.
-2	Bad terminal type.
-3	Bad *capname*.

In addition, if a *capname* is requested for a terminal that has no value for that *capname* (for example, tput -T450 lines), -1 is printed.

SEE ALSO

stty(1), terminfo(4).

NAME

tr — translate characters

SYNOPSIS

tr [-c] [-d] [-s] [*string1* [*string2*]]

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. 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*. Any combination of the flag options -cds may be used.

-c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.

-d Deletes all input characters in *string1*.

-s Squeezes all strings of repeated output characters in *string2* into single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings.

[a-z] Stands for the string of characters whose ASCII codes run from character a to character z, inclusive.

[*a***n*] Stands for *n* repetitions of *a*. If the first digit of *n* is 0, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used, as in the shell, to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

EXAMPLES

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2
```

creates a list of all the words in file1, one per line in file2, where a word is taken to be a maximal string of alphabetics. (The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for 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` flag option, squeezes the newlines to one per occurrence with the `-s` flag option, and directs the output to `file2`.

FILES

`/usr/bin/tr`

SEE ALSO

dd(1), ed(1), sh(1), ascii(5),
"Other Text Processing Tools" in *A/UX Text Processing Tools.*

BUGS

Won't handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

NAME

troff — text formatting and typesetting

SYNOPSIS

troff [-o*list*] [-n*N*] [-s*N*] [-m*name*] [-r*aN*] [-i] [-q] [-a] [-T*dest*] [*file...*]

DESCRIPTION

troff formats text in the named *files* for printing on a photo-typesetter. It is the new "device-independent" version of troff.

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (-) is taken to be a filename corresponding to the standard input. The flag options, which may appear in any order so long as they appear before the files, are

-o*list* Print 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 BUGS, later in this section.)

-n*N* Number the first generated page *N*.

-s*N* Generate 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.

-m*name* Insert the macro file /usr/lib/tmac/tmac.*name* at the beginning of the input *files*.

-r*aN* Set register *a* (one character name) to *N*.

-i Read standard input after the input files are exhausted.

-q Invoke the simultaneous input-output mode of the .rd request.

-a Send a printable ASCII approximation of the results to the standard output.

-T*dest* Prepare 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(1).

The supported typesetter is the Autologic APS-5 (-Taps). For output destined for an Apple Image-

Writer II printer, use the flag `-Tiw` and pipe the output to `daiw`(1). Other output devices may be available.

EXAMPLES

```
troff -o4,8-10 -mabc file1 file2
```

requests formatting of pages 4, 8, 9, and 10 of a document contained in the files named `file1` and `file2` and invokes the macro package `abc`.

FILES

`/bin/troff`	
`/usr/lib/suftab`	suffix hyphenation tables
`/tmp/ta$#`	temporary file
`/tmp/trtmp*`	temporary file
`/usr/lib/tmac/tmac.*`	standard macro files and pointers
`/usr/lib/macros/*`	standard macro files
`/usr/lib/font/dev*/*`	font width tables

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).
"`nroff/troff` Reference Manual" and "Introduction to `troff` and `mm`" in *A/UX Text Processing Tools.*

BUGS

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 *–olist* flag option inside a pipeline (for example, with one or more of `cw`(1), `eqn`(1), and `tbl`(1)), it may cause a harmless `broken pipe` diagnostic if the last page of the document is not specified in *list.*

NAME

true, false — provide 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(1) and/or ksh(1).

EXAMPLES

```
while true
do
        command
done
```

FILES

```
/bin/true
/bin/false
```

SEE ALSO

ksh(1), sh(1).

DIAGNOSTICS

`true` has exit status zero, `false` has exit status nonzero.

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*] [-I] [-m *port*] [-p *type*] [-Q] [-r] [-s] [-S]

reset

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 manual page concerns itself with type initialization, typically done once at login, and flag options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script .profile (for sh(1) users) or .login (for csh(1) 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. To specify which terminal type is usually used on these ports, -m (map) is followed by the appropriate port type identifier, 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.

Bauds are specified as with stty(1), 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. To avoid problems with metacharacters, it is best to place the entire argument to -m (map) within '' characters; users of csh(1) must also put a \ before any ! used here.

Thus,

```
tset -m 'dialup>300:adm3a' -m dialup:dw2 \
-m 'plugboard:?adm3a'
```

causes the terminal type to be set to an adm3a if the port in use is a dialup at a speed greater than 300 baud; to a dw2 if the port is a dialup at a speed of 300 baud or less.

Note: The above command can be entered on one line by omitting the backslash character.

If the *type* above begins with a question mark, 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, default *type* flag option may be given on the command line, not preceded by a -m.

It is often desirable to return the terminal type, as specified by the -m flag options, and information about the terminal to a shell's environment. This can be done using the -s flag option; using the Bourne shell, sh(1),

```
eval `tset -s options... `
```

or using the C shell, csh(1).

```
tset -s options... > tset$$
source tset$$
rm tset$$
```

These commands cause tset to generate as output a sequence of shell commands which place the variables TERM and TERMCAP in the environment; see environ(5).

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 flag 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*).

Other flag options are

`-A` Prompts the user for the terminal type

`-e`*c* Set the erase character to be the named character *c* on all terminals, the default being the backspace character on the terminal, usually CONTROL-H.

`-k`*c* Similar to `-e`, but for the line kill character, rather than the erase character; *c* defaults to CONTROL-X (for purely historical reasons); CONTROL-U is the preferred setting. No kill processing is done if `-k` is not specified.

`-I` Suppresses outputting terminal initialization strings.

`-Q` Suppresses printing the `Erase set to` and `Kill set to` messages.

`-S` Outputs the strings to be assigned to `TERM` and `TERMCAP` in the environment, rather than commands for a shell.

For compatibility with earlier versions of `tset`, a number of flags are accepted whose use is discouraged:

`-d` *type* Equivalent to `-m dialup:`*type*.

`-p` *type* Equivalent to `-m plugboard:`*type*.

`-a` *type* Equivalent to `-m arpanet:`*type*.

`-E`*c* Sets the erase character to *c* only if the terminal can backspace.

`-` Prints the terminal type on the standard output.

`-r` Prints the terminal type on the diagnostic output.

`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

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.

FILES

`/bin/tset`
`/bin/reset`
`/etc/ttytype`
`/etc/termcap`

SEE ALSO

`csh`(1), `sh`(1), `stty`(1), `termcap`(4), `ttytype`(4), `environ`(5).

BUGS

Should be merged with `stty`(1).

`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.

NAME

tsort — topological sort

SYNOPSIS

tsort [*file*]

DESCRIPTION

tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

EXAMPLES

```
ar cr library 'lorder *.o | tsort'
```

intends to build a new library from existing .o files.

FILES

/usr/bin/tsort

SEE ALSO

lorder(1), sort(1), sortbib(1).

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty — get the terminal's name

SYNOPSIS

tty [-l] [-s]

DESCRIPTION

tty prints the path name of the user's terminal. The -l flag option prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line. The -s flag option inhibits printing of the terminal's path name, allowing one to test just the exit code.

EXAMPLES

```
tty
```

produces /dev/tty7 if user is on tty7.

EXIT CODES

2 if invalid options were specified,
0 if standard input is a terminal,
1 otherwise.

DIAGNOSTICS

"not on an active synchronous line" if the standard input is not a synchronous terminal and -l is specified.
"not a tty" if the standard input is not a terminal and -s is not specified.

FILES

/bin/tty

SEE ALSO

ttyname(3C).

See machid(1)

See machid(1)

See machid(1)

See machid(1)

NAME

ucbdiff — differential file and directory comparator

SYNOPSIS

ucbdiff [-l] [-r] [-s] [-S*name*] [-cefhn] [-biwt] *dir1 dir2*
ucbdiff [-cefhn] [-biwt] *file1 file2*
ucbdiff [-D*string*] [-biw] *file1 file2*

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. Options, when comparing directories, are:

-l Long output format; each set of text file differences is piped through pr(1) 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 Causes ucbdiff to report files which are the same and are otherwise not mentioned.

-S*name*
Starts a directory ucbdiff in the middle, beginning with the file *name*.

When 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 >.

Except for -b, -w, -i, or -t (which may be given with any of the others), the following options are mutually exclusive.

-e Produces a script of a, c, and d commands for the editor ed, 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(1) 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.

-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(1).

-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 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.)

`-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.

`-D`*string* Causes `ucbdiff` to create 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*.

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

`-w` Similar to `-b` but causes whitespace (blanks and tabs) to be totally ignored. For example, the following two lines will compare equal.

```
if ( a == b )
if(a==b)
```

`-i` Ignores the case of letters. For example, `A` will compare equal to `a`.

`-t` Will expand 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.

FILES

`/usr/ucb/ucbdiff`
`/tmp/d?????`
`/usr/lib/ucbdiffh` for `-h`
`/bin/diff` for directory comparisons
`/bin/pr`

SEE ALSO

`cc`(1), `cmp`(1), `comm`(1), `diff`(1), `ed`(1), `rcs`(1), `ucbdiff3`(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some, 2 for trouble.

BUGS

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.

NAME

ucbdiff3 — 3-way differential file comparison

SYNOPSIS

ucbdiff3 [-exEX3] *file1 file2 file3*

DESCRIPTION

ucbdiff3 is used by the rcs(1) Revision Control System. ucbdiff3 compares three versions of a file and publishes disagreeing ranges of text flagged with these codes.

====	all three files differ
====1	*file1* is different
====2	*file2* is different
====3	*file3* is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of two ways.

f:*n1*a	Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.
f:*n1*,*n2*c	Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a c indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the -e option, ucbdiff3 publishes a script for the editor ed that will incorporate into *file1* all changes between *file2* and *file3*, (the changes that would normally be flagged ==== and ====3). Option -x (-3) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to *file1*.

```
(cat script; echo '1,$p') | ed - file1
```

The -E and -X are similar to -e and -x, respectively, 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.

For example, suppose lines 7–8 are changed in both `f1` and `f2`. Applying the edit script generated by the command

```
ucbdiff3 -E f1 f2 f3
```

to `f1` results in the file:

lines 1-6 of `f1`
`<<<<<<< f1`
lines 7-8 of `f1`
`=======`
lines 7-8 of `f3`
`>>>>>>> f3`
rest of `f1`

The `-E` option is used by RCS `merge`(1) to insure that overlapping changes in the merged files are preserved and brought to someone's attention.

FILES

`/usr/ucb/ucbdiff3`
`/tmp/d3?????`
`/usr/lib/ucbdiff3`

SEE ALSO

`diff`(1), `diff3`(1), `rcs`(1), `ucbdiff`(1).

BUGS

Text lines that consist of a single period (.) will defeat `-e`.

NAME

ul — filter special underlining sequences imbedded in text for use at a display device

SYNOPSIS

ul [–t *terminal*] [*name* ...]

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 –t is present, *terminal* is used as the terminal kind. Otherwise, first 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. If the terminal can overstrike, or handles underlining automatically, ul behaves like cat(1). If the terminal cannot underline, underlining is ignored.

EXAMPLES

```
ul file1
```

displays file1 on the terminal with underlined portions of the file either underlined, or in reverse video when this flag option is supported for the terminal.

FILES

/usr/bin/ul
/bin/cat
/usr/lib/terminfo

SEE ALSO

colcrt(1), man(1), nroff(1).

BUGS

nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

NAME

uname — display identification information about the current system

SYNOPSIS

uname [-a] [-m] [-n] [-r] [-s] [-v]

DESCRIPTION

uname displays the system name of the current system on the standard output file. It is mainly useful to determine which system one is using. The flag options cause selected information returned by uname(2) to be displayed:

-s display the system name (default).

-n display the nodename (the nodename may be a name that the system is known by to a communications network).

-r display the operating system release.

-v display the operating system version.

-m display the machine hardware name.

-a display all the above information.

EXAMPLES

```
uname
```

on A/UX would display on the screen

```
A/UX
```

FILES

/bin/uname

SEE ALSO

uname(2).

See compact(1)

See compress(1)

See expand(1)

NAME

unget — undo a previous get of an SCCS file

SYNOPSIS

unget [-n] [-r*SID*] [-s] *file...*

DESCRIPTION

unget undoes the effect of a get -e done prior to creating the intended new delta. If a directory is named, unget behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

-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 keyletter is necessary only if two or more outstanding gets for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

-s Suppresses the printout, on the standard output, of the intended delta's *SID*.

-n Causes the retention of the gotten file which would normally be removed from the current directory.

EXAMPLES

```
% unget s.test1.c
1.2
```

undoes version 1.2 of test1.c set up for editing by an earlier get -e.

FILES

/usr/bin/unget

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

DIAGNOSTICS

Use `help`(1) for explanations.

NAME

uniq — report repeated lines in a file

SYNOPSIS

uniq [-u] [-d] [-c] [+*n*] [-*n*] [*input* [*output*]]

DESCRIPTION

uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see sort(1). If the -u flag option is used, just the lines that are not repeated in the original file are output. The -d flag option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the -u and -d mode outputs.

The -c flag option supersedes -u and -d and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

-*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of nonspace, nontab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

EXAMPLES

```
uniq file1
```

prints contents of file1 with adjacent identical lines removed.

FILES

/usr/bin/uniq

SEE ALSO

comm(1), diff(1), sort(1).

NAME

units — conversion program

SYNOPSIS

```
units
```

DESCRIPTION

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively (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

pound is not recognized as a unit of mass; lb 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
    /usr/lib/unittab
```

unpack(1)

See pack(1)

NAME

`updater` — update files between two machines

SYNOPSIS

`updater` [*key*] *local remote* ...

DESCRIPTION

`updater` updates files between two machines.

One of the following *key* letters must be included:

- `t` Take files from the remote machine, updating the local machine.
- `p` Put files from the local machine onto the remote machine, updating the remote machine.
- `d` List the difference between files on the local and remote machines.

The following key letters are optional:

- `u` Update a file only if it exists on both machines; this is the default condition.
- `r` Replace a file if it did not exist on the destination machine.

local refers to the local directory name.

remote refers to the remote directory names. Only one remote name can be specified if the `p` (put) key is specified.

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

```
cd remote ; sumdir . | sort +2 > /tmp/rXXXXX
```

to remote machine for each remote system;

```
cd local ; sumdir . | sort > /tmp/lXXXXX
```

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

```
updater d...
```

uses `/dev/tty0` to communicate with a remote machine and compares directories on the remote and local systems.

FILES

`/usr/bin/updater`

NAME

uptime — show 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(1) command.

FILES

/usr/ucb/uptime
/dev/kmem
/etc/utmp

SEE ALSO

ps(1), ruptime(1N), w(1).

NAME

users — compact list of users who are on the system

SYNOPSIS

users [*file*]

DESCRIPTION

users lists the login names of the users currently on the system in a compact, one-line format. users takes the file from which it reads user information an optional argument; the default is /etc/utmp.

FILES

/usr/ucb/users
/etc/utmp

SEE ALSO

finger(1), w(1), who(1).

NAME

uucp, uulog, uuname — UNIX® system to UNIX system copy

SYNOPSIS

[-c] [-C] [-d] [-e*sys*] [-f] [-j] [-m*file*] [-n*user*] [-r] *source-files destination-file*

uulog [-s*sys*] [-u*user*]

uuname [-l] [-v]

DESCRIPTION

uucp

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A filename may be a pathname on your system, or may have the form

system-name ! *pathname*

where *system-name* is taken from a list of system names that uucp knows about. In addition, *system-name* may be a list of names such as

system-name ! *system-name* ! . . . ! *system-name* ! *pathname*

in which case an attempt is made to send the file via the specified route and only to a destination in the directory PUBDIR (see below). Care should be taken to ensure that intermediate nodes in the route are willing to forward information.

The shell metacharacters ?, *, and [...] appearing in *pathname* are expanded on the appropriate system.

Pathnames may be any of the following:

a full pathname

a pathname preceded by
~*user* where *user* is a login name on the specified system and is replaced by that user's login directory

a pathname preceded by
~/*user* where *user* is a login name on the specified system and is replaced by that user's directory under PUBDIR, where PUBDIR is the public directory on the remote system to which all uucp users have access and at which they enter

anything else if
uucp prefixes it with the current directory

If the result is an erroneous pathname for the remote system, the copy fails. If the *destination-file* is a directory, the last part of the *source-file* name is used.

If a simple *~user* destination is inaccessible to uucp, data is copied to a spool directory, and the user is notified by mail(1).

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions. See chmod(2).

The following flag options are interpreted by uucp:

-d Make all necessary directories for the file copy (default).

-f Do not make intermediate directories for the file copy.

-c Use the source file when copying out rather than copying the file to the spool directory (default).

-C Copy the source file to the spool directory.

-m*file* Report the status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

-n*user* Notify *user* on the remote system that a file was sent.

-e*sys* Send the uucp command to system *sys* to be executed there.

Note: This will be successful only if the remote machine allows uucp to be executed by /usr/lib/uucp/uuxqt.

-g Assign a priority, or grade, to the uucp request, so uucico can later process requests of selected grades only.

-r Queue the job, but do not start the file-transfer process. By default, a file-transfer process is started each time uucp is evoked.

-j Control the writing of the uucp job number to standard output (see below).

uucp associates a job number with each request. This job number can be used by uustat to obtain the status of the job or terminate the job.

The environment variable JOBNO and the -j flag option are used to control the listing of the uucp job number on standard output. If the environment variable JOBNO is undefined or set to OFF, the job number is not listed (default). If uucp is then invoked with the -j flag option, the job number is listed. If the environment variable JOBNO is set to ON and is exported, a job number is written to standard output each time uucp is invoked. In this case, the -j flag option suppresses the output of the job number.

uulog

uulog queries a summary log of uucp and uux(1C) transactions in the file /usr/spool/uucp/LOGFILE.

The following flag options cause uulog to print logging information:

-s*sys* Print information about work involving system *sys*. If *sys* is not specified, then logging information for all systems is printed.

-u*user* Print information about work done for the specified *user*. If *user* is not specified, then logging information for all users is printed.

uuname

uuname lists the uucp names of known systems. The -l flag option returns the local system name. The -v flag option prints additional information about each system. A description is printed for each system that has a line of information in /usr/lib/uucp/ADMIN. The format of ADMIN is:

sysname TAB *description* TAB

EXAMPLES

```
uucp file1 ucbvax! /usr/spool/uucppublic/file2
```

sends file1 from the local machine via the uucp network to the ucbvax machine, where it is saved as the file /usr/spool/uucppublic/file2.

FILES

/usr/bin/uucp	
/usr/bin/uulog	
/usr/bin/uuname	
/usr/lib/uudemon.day	Performs once per day
/usr/lib/uudemon.hour	Performs once per hour
/usr/lib/uudemon.week	Performs once per week

`/usr/spool/uucp`	Spool directory
`/usr/spool/uucppublic`	Public directory (`PUBDIR`) for receiving and sending
`/usr/lib/uucp/*`	Other data and program files

NOTES

To send files that begin with a dot, for example, `.profile`, the files must be qualified with the dot. For example, `.prof*`, `.profile`, `.profil?` are correct, whereas `*prof*`, `?profile` are incorrect.

`uucp` does not generate a job number for a strictly local transaction.

SEE ALSO

`ftp`(1N), `mail`(1), `telnet`(1N), `tip`(1C), `uux`(1C), `chgnod`(1M), `chmod`(2),

"Using UUCP," in the *AIUX Communications User's Guide*.

WARNINGS

The domain of remotely accessible files may (and for obvious security reasons, usually should) be severely restricted. Very likely, you will not be able to fetch files by pathname, so ask a responsible person on the remote system to send them to you. For the same reasons, you will probably not be able to send files to arbitrary pathnames. As distributed, the remotely accessible files are those whose names begin `/usr/spool/uucppublic` (equivalent to `~nuucp` or just `~`).

BUGS

All files received by `uucp` are owned by `uucp`.

The `-m` flag option only works when sending files or receiving a single file. Receiving multiple files specified by the special shell characters `?`, `*`, and `[...]` does not activate the `-m` flag option.

The `-m` flag option does not work if all transactions are local or if `uucp` is executed remotely via the `-e` flag option.

The `-n` flag option functions only when the source and destination are not on the same system.

Only the first six characters of *system-name* are significant. Any excess characters are ignored.

See uuencode(1C)

NAME

uuencode, uudecode — encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [*source*] *remotedest*

uudecode [*file*]

DESCRIPTION

uuencode and uudecode are used to send a binary file via uucp (or other) mail. This combination may be used over indirect mail links even when uusend(1C) is not available.

uuencode takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the *remotedest* for recreation on the remote system.

uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user decode should be filtered through the uudecode program. This way the file is created automatically without human intervention. This is possible on the uucp network either by using sendmail or by making rmail a link to mailx instead of mail. In each case, an alias must be created in a master file to get the automatic invocation of uudecode.

If these facilities are not available, the file may be sent to a user on the remote machine who can uudecode it manually.

The encode file has an ordinary text form and may be edited by any text editor to change the mode or remote name.

FILES

/usr/ucb/uudecode
/usr/ucb/uuencode

SEE ALSO

mail(1), uucp(1C), uusend(1C), uux(1C).
"Using UUCP" in *A/UX Communications User's Guide.*

BUGS

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking `uudecode` (often `uucp`) must have write permission on the specified file.

See uucp(1C)

See uucp(1C)

See uuto(1C)

NAME

uusend — send a file to a remote host

SYNOPSIS

uusend [-m*mode*] *sourcefile sys1* ! *sys2* ! . . . ! *remotefile*

DESCRIPTION

uusend sends a file to a given location on a remote system. The system need not be directly connected to the local system, but a chain of uucp(1C) links needs to connect the two systems.

If the -m flag option is specified, the mode of the file on the remote end will be taken from the octal number given. Otherwise, the mode of the input file will be used.

The *sourcefile* can be -, meaning to use the standard input. Both of these flag options are primarily intended for internal use of uusend.

The *remotefile* can include the ~*userid* syntax.

DIAGNOSTICS

If anything goes wrong any further away than the first system down the line, you will never hear about it.

FILES

/usr/ucb/uusend

SEE ALSO

uucp(1C), uuencode(1), uux(1C).
"Using UUCP" in *A/UX Communications User's Guide*.

BUGS

This command shouldn't exist, since uucp should handle it.

All systems along the line must have the uusend command available and allow remote execution of it.

Some uucp systems have a bug where binary files cannot be the input to a uux command. If this bug exists in any system along the line, the file will show up severely munged.

NAME

uustat — uucp status inquiry and job control

SYNOPSIS

uustat [-c*hour*] [-j*jobn*] [-k*jobn*] [-m*mch*] [-M*mch*] [-o*hour*] [-O] [-q] [-r*jobn*] [-s*sys*] [-u*user*] [-y*hour*]

DESCRIPTION

uustat will display the status of, or cancel, previously specified uucp commands, or provide general status on uucp connections to other systems. The following flag options are recognized:

-j*jobn* Report the status of the uucp request *jobn*. If all is used for *jobn*, the status of all uucp requests is reported. An argument must be supplied; otherwise the usage message will be printed and the request will fail.

-k*jobn* Kill the uucp request whose job number is *jobn*. The killed uucp request must belong to the person issuing the uustat command unless one is the superuser.

-r*jobn* Rejuvenate *jobn*. That is, *jobn* is touched so that its modification time is set to the current time. This prevents uuclean from deleting the job until the jobs modification time reaches the limit imposed by uuclean.

-c*hour* Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user uucp or the superuser.

-u*user* Report the status of all uucp requests issued by *user*.

-s*sys* Report the status of all uucp requests which communicate with remote system *sys*.

-o*hour* Report the status of all uucp requests which are older than *hour* hours.

-y*hour* Report the status of all uucp requests which are younger than *hour* hours.

-m*mch* Report the status of accessibility of machine *mch*. If *mch* is specified as all, then the status of all machines known to the local uucp are provided.

-M*mch* This is the same as the -m option except that two times are printed. The time that the last status was obtained and the time that the last successful transfer to that system occurred.

-O Report the uucp status using the octal status codes listed below. If this option is not specified, the verbose description is printed with each uucp request.

-q List the number of jobs and other control files queued for each machine and the time of the oldest and youngest file queued for each machine. If a lock file exists for that system, its date of creation is listed.

When no options are given, `uustat` outputs the status of all `uucp` requests issued by the current user. Note that only one of the flag options `-j`, `-m`, `-k`, `-c`, `-r`, can be used with the rest of the other options.

For example, the command:

```
uustat -uhdc -smhtsa -y72
```

will print the status of all `uucp` requests that were issued by user `hdc` to communicate with system `mhtsa` within the last 72 hours. The meanings of the job request status are:

job-number user remote-system command-time status-time status

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
000001	the copy failed, but the reason cannot be determined
000002	permission to access local file is denied
000004	permission to access remote file is denied
000010	bad `uucp` command is generated
000020	remote system cannot create temporary file
000040	cannot copy to remote directory
000100	cannot copy to local directory
000200	local system cannot create temporary file
000400	cannot execute `uucp`
001000	copy (partially) succeeded
002000	copy finished, job deleted
004000	job is queued
010000	job killed (incomplete)
020000	job killed (complete)

The meanings of the machine accessibility status are:

system-name time status

where *time* is the latest status time and *status* is a self-explanatory description of the machine status.

FILES

`/usr/bin/uustat`	
`/usr/spool/uucp`	spool directory
`/usr/lib/uucp/L_stat`	system status file
`/usr/lib/uucp/R_stat`	request status file

SEE ALSO

uucp(1C).

"The UUCP System" in *A/UX Local System Administration.*

NAME

uuto, uupick — public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [–m] [–p] *source-files destination*
uupick [–s*system*]

DESCRIPTION

uuto sends *source-files* to *destination*. uuto uses the uucp(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!*user*

where *system* is taken from a list of system names that uucp knows about (see uuname). *logname* is the login name of someone on the specified system.

Two flag options are available:

–p	Copy the source file into the spool directory before transmission.
–m	Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the uucp source. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/*files*.

The destined recipient is notified by mail(1) of the arrival of files.

uupick accepts or rejects the files transmitted to the user. Specifically, uupick searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from-system : [*file file-name*] [*dir dirname*] ?

uupick then reads a line from the standard input to determine the disposition of the file:

newline	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir* (current directory is default).

a [*dir*]	Same as m except moving all the files sent from *system*.
p	Print the content of the file.
q	Stop.
EOT (CONTROL-D)	Same as q.
! *command*	Escape to the shell to do *command*.
*	Print a command summary.

uupick invoked with the −s*system* option will only search the PUBDIR for files sent from *system*.

EXAMPLES

```
uuto -p file1 file2 file3 ucbvax!kalash
```

would send the three files to "kalash" on "ucbvax"

```
uupick [executed by user kalash]
```

would tell him what has arrived and from where.

NOTES

In order to send any files, they must be readable by others; see In order to send any files, they must be readable by others; see chmod(1). chmod(1). In order to send files that begin with a dot (for example, .profile) the file must be qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

FILES

```
/usr/bin/uupick
/usr/bin/uuto
PUBDIR=/usr/spool/uucppublic
```

SEE ALSO

chmod(1), mail(1), uucp(1C), uustat(1C), uux(1C), uuclean(1M).
"Using UUCP" in *A/UX Communications User's Guide.*

NAME

uux — UNIX-to-UNIX system command execution

SYNOPSIS

uux [-] [-g*grade*] [-j] [-l] [-m*file*] [-n] [-p] [-r] [-x*level*] [-z] *command-string*

DESCRIPTION

uux will gather zero or more files from various systems, execute a command on a specified system, and then send standard output to a file on a specified system. Note that for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from uux. Many sites will permit little more than the receipt of mail (see mail(1)) via uux.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and filenames may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

Filenames may be one of the following:

(1) A full pathname.

(2) A pathname preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory.

(3) Anything else is prefixed by the current directory.

Any special shell characters such as <>; | should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

uux will attempt to get all files to the execution system. For files which are output files, the filename must be escaped by using parentheses.

uux will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine. Executable commands are listed in /usr/lib/uucp/L.cmds on the remote system. The format of the L.cmds file is

cmd, *machine1*, *machine2*, ...

If no machines are specified, then any machine can execute *cmd*. If machines are specified, only the listed machines can execute *cmd*. If the desired command is not listed in L.cmds, then no

machine can execute that command.

Redirection of standard input and output is usually restricted to files in PUBDIR. Directories into which redirection is allowed must be specified in /usr/lib/uucp/USERFILE by the system administrator. See "Using UUCP" in *A/UX Communications User's Guide*.

The following flag options are interpreted by uux.

- The standard input to uux is made the standard input to the *command-string*.

-g*grade*
Sets transmit priority to the level *grade*. The argument *grade* is a single letter or number. Lower ASCII-sequence characters will cause the job to be transmitted earlier during a particular conversation.

-j Controls writing of the uucp job number to standard output.

-l Tries to make a link from the original file to the spool directory. If the link cannot be made, then copy the file.

-m*file* Reports the status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

-n Sends no notification to user.

-p Takes the input for the *command-string* from a pipe; the same as -.

-r Queues the job, but does not start the file transfer.

-x*level*
Produces debugging output on the standard output. The *level* is a number between 0 and 9; higher numbers give more information than lower numbers.

-z Notifies the user if the command succeeds.

uux associates a job number with each request. This job number can be used by uustat to obtain status or terminate the job.

The environment variable JOBNO and the -j flag option are used to control the listing of the uux job number on standard output. If the environment variable JOBNO is undefined or set to OFF, the job number will not be listed (default). If uucp is then invoked with the -j flag option, the job number will be listed. If the environment variable JOBNO is set to ON and is exported, a job

number will be written to standard output each time uux is invoked. In this case, the -j flag option will suppress output of the job number.

EXAMPLES

```
uux !diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff
```

will get the f1 files from the usg and pwba machines, execute a diff command, and put the results in f1.diff in the local directory.

```
uux a!uucp b!/usr/file \(c!/usr/file\)
```

will send a uucp command to system a to get /usr/file from system b and send it to system c.

FILES

/usr/bin/uux	
/usr/lib/uucp/L.sys	list of system names and when to call them
/usr/lib/uucp/L.cmds	list of commands for uuxqt to execute
/usr/lib/uucp/L-devices	list of device codes and speeds
/usr/lib/uucp/L-dialcodes	list of phone numbers in L.sys
/usr/lib/uucp/SYSTEMNAME	name of this system
/usr/lib/uucp/USERFILE	list of users and required pathname prefixes
/usr/lib/uucp/uucico	copy in, copy out program; called by uucp
/usr/lib/uucp/uuclean	spool directory cleanup program; called by uucp
/usr/lib/uucp/uuxqt	command execution program; called by uucp
/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory (PUBDIR)

SEE ALSO

uucp(1C), mail(1), uuclean(1M).
"Using UUCP" in *A/UX Communications User's Guide*.

BUGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.

The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

Only the first six characters of the *system-name* are significant; excess characters are ignored.

NAME

val — validate SCCS file

SYNOPSIS

```
val -
val [-mname] [-rSID] [-s] [-ytype] 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. The arguments consist of keyletter arguments, which begin with a –, and named files.

val has a special argument, –, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

- **-s** The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.
- **-r*SID*** The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* 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.
- **-m*name*** The argument value *name* is compared with the SCCS %M% keyword in *file*.
- **-y*type*** The argument value *type* is compared with the SCCS %Y% keyword in *file*.

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.

FILES

`/usr/bin/val`

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

DIAGNOSTICS

Use `help`(1) for explanations.

BUGS

`val` can process up to 50 files on a single command line. Any number above 50 will produce a `core` dump.

See machid(1)

NAME

vc — version control

SYNOPSIS

vc [-a] [-c*char*] [-s] [-t] [*keyword=value*]...

DESCRIPTION

The vc command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as vc command arguments.

A control statement is a single line beginning with a control character, (unless the -t keyletter is used, in which case, all characters up to and including the first tab are ignored , and what follows begins the control statement (see below)). The default control character is colon (:) . This can be changed by the -c keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a noncontrol character are copied in their entirety.

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(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The -a keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter Arguments

-a Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in vc statements.

`-t` All characters from the beginning of a line up to and including the first tab character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the tabs are discarded.

`-c`*char* Specifies a control character to be used in place of `:`.

`-s` Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements

`:dcl` *keyword* [, ..., *keyword*]
Used to declare keywords. All keywords must be declared.

`:asg` *keyword=value*
Used to assign values to keywords. An `asg` statement overrides the assignment for the corresponding keyword on the `vc` command line and all previous `asg`'s for that keyword. Keywords declared, but not assigned values have null values.

`:if` *condition*
:
:

`:end`
Used to skip lines of the standard input. If the condition is true all lines between the `if` statement and the matching `end` statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening `if` statements and matching `end` statements are recognized solely for the purpose of maintaining the proper `if-end` matching.

The syntax of a condition is:

```
<cond>  ::= [ "not" ] <or>
<or>    ::= <and> | <and> "|" <or>
<and>   ::= <exp> | <exp> "&" <and>
<exp>   ::= "(" <or> ")" | <value> <op>
            <value>
<op>    ::= "=" | "!=" | "<" | ">"
<value> ::= <arbitrary ASCII string> | <numeric
            string>
```

The available operators and their meanings are:

`=` equal

`!=`	not equal
`&`	and
`|`	or
`>`	greater than
`<`	less than
`( )`	used for logical groupings
`not`	may only occur immediately after the `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*
Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the `-a` keyletter.

`:on`

`:off`
Turn on or off keyword replacement on all lines.

`:ctl` *char*
Change the control character to char.

`:msg` *message*
Prints the given message on the diagnostic output.

`:err` *message*
Prints the given message followed by:

```
ERROR: err statement on line ... (915)
```

on the diagnostic output. vc halts execution, and returns an exit code of 1.

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.
```

FILES

/usr/bin/vc

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

DIAGNOSTICS

Use help(1) for explanations.

EXIT CODES

0 – normal
1 – any error

See vi(1)

NAME

version — reports version number of files

SYNOPSIS

version *name...*

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 no version number is associated with the file, it reports:

```
No version header
```

version also reports the object file format of the file, i.e., either

```
Coff object file format
```

or

```
Old a.out object file format.
```

version 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

SEE ALSO

strings(1), what(1).

NAME

vi, view, vedit — 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...*

DESCRIPTION

vi (visual) is a display-oriented text editor based on an underlying line editor ex(1). It is possible to use the command mode of ex from within vi and vice versa.

When using vi, changes you make to the file are reflected by what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

FLAG OPTIONS

The following flag options are interpreted by vi.

-l	LISP mode; indents appropriately for lisp code, the (), {}, [[, and]] commands in vi and *open* are modified to have meaning for *lisp*.
-r[*file*]	Recover *file* after an editor or system crash. If *file* is not specified, a list of all saved files will be printed.
-R	Read only mode; the readonly flag is set, preventing accidental overwriting of the file.
-t *tag*	Edits the file containing the *tag* and position 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	Encryption mode; a key is prompted for allowing creation or editing of an encrypted file. This encryption scheme is not secure.
+*command*	The specified ex command is interpreted before editing begins.

The *name* argument indicates files to be edited.

The `view` invocation is the same as `vi` except that the `readonly` flag is set.

The `vedit` invocation is intended for beginners. The `report` flag is set to 1, and the `showmode` and `novice` flags are set. These defaults make it easier to initially learn the editor.

VI 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 `a i A I o O c C s S 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.

COMMAND SUMMARY

Sample Commands

←↓↑→	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`	. . . 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, Canceling

ESCAPE	end insert or incomplete *cmd*
interrupt	your interrupt
CONTROL-L	reprints screen
CONTROL-R	reprint 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*
:w!*name*RETURN	overwrite file *name*
:shRETURN	run shell, then return
:!*cmd*RETURN	run *cmd*, then return
:nRETURN	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*
^]	:ta, following word is *tag*

In general, any ex or ed command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a RETURN.

Positioning Within 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	... at center
/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 non-white
RETURN	return, same as +
↓ or j	next line, same column
↑ or k	previous line, same column

Character Positioning

^	first nonwhite
0	beginning of line
$	end of line
h or →	forward
l or ←	backward
CONTROL-H	same as ←
space	same as →
f*x*	find *x* forward
F*x*	f backward
t*x*	up to *x* forward
T*x*	back up to *x*
;	repeat last f F t or T
,	inverse of ;
|	to specified column
%	find matching ({) or }

Words, Sentences, 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, Retrieve

`u`	undo last change
`U`	restore current line
`.`	repeat last change
`"`*d*`p`	retrieve *d*th last delete

FILES

```
/usr/bin/vi
/usr/bin/view
/usr/bin/vedit
```

SEE ALSO

ex(1).

"Using `vi`" in the *A/UX Text Editing Tools.*

BUGS

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.

See vi(1)

NAME

w — who is on and what they are doing?

SYNOPSIS

w [-h] [-u] [-s] [-l] [*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.

The -h flag option suppresses the heading. The -u flag option suppresses everything but the heading, as in uptime(1). The -s flag option 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. -l gives the long output, which is the default.

If a *user* name is included, the output will be restricted to that user.

FILES

/usr/ucb/w
/etc/utmp
/dev/kmem

SEE ALSO

who(1), finger(1), ps(1), uptime(1), users(1), utmp(1).

BUGS

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.

`w` does not know about the new conventions for detection of background jobs. It will sometimes find a background job instead of the right one.

NAME

wc — word count

SYNOPSIS

wc [-c] [-l] [-w] [*name...*]

DESCRIPTION

wc counts lines, words and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or newlines.

The flag options l, w, and c may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is -lwc.

When *names* are specified on the command line, they will be printed along with the counts.

EXAMPLES

```
wc filea fileb filec
```

reports the number of lines, words, and characters in each of the files, as well as the totals.

FILES

/bin/wc

SEE ALSO

sum(1), sumdir(1).

NAME

what — identify SCCS files

SYNOPSIS

what [-s] *file* ...

DESCRIPTION

what searches the given files for all occurrences of the pattern that get(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ", >, 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
```

what is intended to be used in conjunction with the SCCS command get(1), which inserts identifying information automatically, but it can also be used where the information is inserted manually. Only one flag option exists:

-s Quit after finding the first occurrence of pattern in each file.

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
```

FILES

/usr/bin/what

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

DIAGNOSTICS

Exit status is 0 if any matches are found, otherwise 1. Use help(1) for explanations.

BUGS

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

NAME

whatis — display a brief description for the named manual page entry

SYNOPSIS

whatis *command* ...

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. If the line starts with

name (*section*) ...

you may type

man *section name*

to get the documentation for it. For instance, if you type

```
whatis ed
```

you will see

```
ed, red (1)          - text editor
```

This tells you that the *section* for ed is 1. To see the manual entry for ed, you may then type

```
man 1 ed
```

and the entire entry will be printed.

FILES

/usr/ucb/whatis
/usr/lib/whatis

SEE ALSO

apropos(1), man(1), whereis(1), which(1).

NAME

whereis — locate source, binary, and online help file for a command

SYNOPSIS

whereis [-b] [-m] [-s] [-u] [-B *dir* [-f]] [-M *dir* [-f]] [-S *dir* [-f]] *name* ...

DESCRIPTION

whereis locates source/binary and manual sections for specified files. The supplied names are first stripped of leading pathname components and standard extensions for source files and manual entries, for example, .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. If any of the -b, -s, or -m flags are given, then whereis searches only for binaries, sources, or manual sections respectively (or any two thereof). The -u flag may be used to search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus whereis -m -u * asks for those files in the current directory which have no documentation.

Finally, the -B, -M, and -S flags may be used to change or otherwise limit the places where whereis searches. The -f file flag is used to terminate the last such directory list and signal the start of file names.

EXAMPLES

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 *
```

FILES

```
/usr/bin/whereis
/usr/src/*
/usr/catman/man/*
/bin
/lib
/etc
/usr/bin
/usr/lib
/usr/etc
/usr/ucb
```

SEE ALSO

whatis(1), which(1).

BUGS

Since the program uses chdir(2) to run faster, pathnames given with the -M, -S, and -B must be full; that is, they must begin with a /.

NAME

which — display the directory path to a file by interpreting PATH and alias settings

SYNOPSIS

which [*name...*]

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.

Note: which(1) Operates only with csh(1).

FILES

/usr/ucb/which	
~/.cshrc	source of aliases and path values

SEE ALSO

whereis(1), which(1).

DIAGNOSTICS

A diagnostic 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.

NAME

who — who is on the system?

SYNOPSIS

who [-a] [-b] [-d] [-H] [-l] [-p] [-q] [-r] [-s] [-t] [-T] [-u] [*file*]

who am i

who am I

DESCRIPTION

who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process ID of the command interpreter (shell) for each current 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.

who with am i or am I added on identifies the invoking user.

Except for the default -s flag option, the general format for output entries is:

name[*state*] *line time activity pid* [*comment*] [*exit*]

With flag options, who can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the init process. These flag options are:

-a This flag option processes /etc/utmp or the named *file* with all options turned on.

-b This flag option indicates the time and date of the last reboot.

-d This flag option displays all processes that have expired and not been respawned by init.

-H This flag option will print column headings above the regular output.

-l This flag option lists only those lines on which the system is waiting for someone to login. The *name* field is LOGIN in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

-p This flag option lists any non-getty process which is currently active and has been previously spawned by init. The *name* field is the name of the program executed by init as found in /etc/inittab. The *state*, *line*, and *activity*

fields have no meaning. The *comment* field shows the *id* field of the line from `/etc/inittab` that spawned this process. See `inittab`(4). The *exit* field appears for dead processes and contains the termination and exit values (as returned by `wait`(2)), of the dead process. This can be useful in determining why a process terminated.

`-q` This is a quick *who*, displaying only the names and the number of users currently logged on. When this flag option is used, all other flag options are ignored.

`-r` This flag option indicates the current *run-level* of the `init` process.

`-s` This flag option is the default and lists only the *name*, *line*, *time*, and *remote host* (if any) fields.

`-t` This flag option indicates the last change to the system clock (via the `date`(1) command) by `root`. See `su`(1).

`-T` This flag option is the same as the `-u` flag option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a – appears if it is not. `root` can write to all lines having a + or a – in the *state* field. If a bad line is encountered, a `?` is printed.

`-u` This flag option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory `/dev`. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked `old`. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process ID of the user's shell. The *comment* is the comment field associated with this line as found in `/etc/inittab` (see `inittab`(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.

EXAMPLES

```
who am i
```

reports the name under which you are currently logged in.

FILES

```
/bin/who
/etc/utmp
/etc/wtmp
/etc/inittab
```

SEE ALSO

date(1), login(1), mesg(1), ps(1), su(1), users(1), w(1), whoami(1), init(1M), wait(2), inittab(4), utmp(4).

NAME

whoami — print 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
/etc/passwd Name data base

SEE ALSO

id(1), who(1).

NAME

write — write to another user

SYNOPSIS

write *user* [*line*]

DESCRIPTION

write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from *yourname* (tty??) [*date*] ...

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent, or the recipient has executed "mesg n". At that point write writes EOT on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument can be used to indicate which line or terminal to send to (e.g., tty00); otherwise, the first writable instance of the user found in /etc/utmp is assumed and the following message posted:

user is logged on more than one place.
You are connected to "*terminal*".
Other locations are:
terminal

Permission to write may be denied or granted by use of the mesg(1) command. Writing to others is normally allowed by default. Certain commands, in particular nroff(1) and pr(1) disallow messages in order to prevent interference with their output. However, if the user has 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.

The following protocol is suggested for using write: when you first write to another user, wait for them to write back before starting to send. Each person should end a message with a distinctive signal (i.e., (o) for "over") so that the other person knows when to reply. The signal (oo) (for "over and out") is suggest-

ed when conversation is to be terminated.

EXAMPLES

```
write judy tty7
```

writes to user "judy" on terminal 7, unless messages have been refused with mesg(1).

FILES

/bin/write
/etc/utmp
/bin/sh

SEE ALSO

mail(1), mesg(1), nroff(1), pr(1), sh(1), talk(1N), who(1), wall(1M).

DIAGNOSTICS

`user is not logged in`	if the person you are trying to `write` to is not logged in.
`Permission denied`	if the person you are trying to `write` to denies that permission (with `mesg`).
`Warning: cannot respond, set mesg -y`	if your terminal is set to `mesg n` and the recipient cannot respond to you.
`Can no longer write to user`	if the recipient has denied permission (`mesg n`) after you had started writing.

NAME

xargs — construct argument list and execute command

SYNOPSIS

xargs [*flags*] [*command* [*initial-arguments*]]

DESCRIPTION

xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

command, which may be a shell file, is searched for, using one's $PATH. If *command* is omitted, /bin/echo is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or 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 the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see -i flag option). Flag options -i, -l, and -n determine how arguments are selected for each command invocation. When none of these flag options are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated arguments. This process is repeated until there are no more arguments. When there are flag option conflicts (for example, -l versus -n), the last flag option has precedence. Flag option values are:

-l*number* *command* is executed 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. Flag option -x is forced.

`-i`*replstr* Insert mode: *command* is executed for each line from standard input, taking the entire line as a single argument, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and flag option `-x` is also forced. `{}` is assumed for *replstr* if not specified.

`-n`*number* Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If flag option `-x` is also coded, each *number* arguments must fit in the *size* limitation, else `xargs` terminates execution.

`-t` Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

`-p` Prompt mode: The user is asked whether to execute *command* 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*.

`-x` Causes `xargs` to terminate if any argument list would be greater than *size* characters; `-x` is forced by the flag options `-i` and `-l`. When none of the flag options `-i`, `-l`, or `-n` are coded, the total length of all arguments must be within the *size* limit.

`-s`*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If `-s` is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

-e*eofstr* *eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical EOF string if -e is not coded. The value -e with no *eofstr* coded turns off the logical EOF string capability (underbar is taken literally). xargs reads standard input until either end-of-file or the logical EOF string is encountered.

xargs will terminate if either:

it cannot execute *command*

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

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

will move all files from directory $1 to directory $2, and echo each move command just before doing it.

```
(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.

```
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.

```
echo $* | xargs -n2 diff
```

will execute diff(1) with successive pairs of arguments originally typed as shell arguments.

FILES

/usr/bin/xargs

SEE ALSO

csh(1), ksh(1), sh(1).

NAME

xstr — extract strings from C programs to implement shared strings

SYNOPSIS

xstr [-] [-c] [*file*]

DESCRIPTION

xstr maintains a file *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.

The command

xstr -c *name*

will extract the strings from the C source in name, replacing string references by expressions of the form (&xstr[*number*]) for some number. An appropriate declaration of xstr is prefixed to the file. The resulting C text is placed in the file x.c, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

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.

xstr can also be used on a single file. A command

xstr *name*

creates files x.c and xs.c as before, without using or affecting any *strings* file in the same directory.

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. xstr reads from its standard input when the argument - is given. An appropriate command sequence for running xstr after the C preprocessor is:

```
cc -E name.c | xstr -c -
cc -c x.c
mv x.o name.o
```

xstr does not touch the file *strings* unless new items are added, thus make can avoid remaking xs.o unless truly necessary.

FILES

/usr/bin/xstr	
strings	Data base of strings
x.c	Massaged C source
xs.c	C source for definition of array xstr
/tmp/xs*	Temp file when "xstr *name*" doesn't touch strings

SEE ALSO

mkstr(1), strings(1).

BUGS

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.

NAME

yacc — yet another compiler-compiler

SYNOPSIS

yacc [-d] [-l] [-t] [-v] *grammar*

DESCRIPTION

yacc converts a context-free grammar into a set of tables for a simple automaton which executes an lr(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, y.tab.c, must be compiled by the C compiler to produce a program 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; lex(1) is useful for creating lexical analyzers usable by yacc.

If the -v flag is given, the file y.output is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the -d flag is used, the file y.tab.h is generated with the #define statements that associate the yacc-assigned "token codes" with the user-declared "token names" This allows source files other than y.tab.c to access the token codes.

If the -l flag is given, the code produced in y.tab.c will *not* contain any #line constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in y.tab.c under conditional compilation control. By default, this code is not included when y.tab.c is compiled. However, when yacc's -t flag option is used, this debugging code will be compiled by default. Independent of whether the -t flag option was used, the runtime debugging code is under the control of YYDEBUG, a preprocessor 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

```
yacc file1.y
```

invokes `yacc` to process file `file1.y` in `yacc`-format.

FILES

`/bin/yacc`
`y.output`
`y.tab.c`
`y.tab.h`
`yacc.tmp`,`yacc.debug`,`yacc.acts` (temporary files)
`/usr/lib/yaccpar`

SEE ALSO

`lex`(1), `malloc`(3X).
"`yacc` Refrence" in *A/UX Programming Languages and Tools, Volume 2.*

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the `y.output` file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS

Because file names are fixed, at most one `yacc` process can be active in a given directory at a time.

NAME

yes — generate y entries in response to requests for input

SYNOPSIS

yes [*expletive*]

DESCRIPTION

yes repeatedly outputs "y" or if *expletive* is given, that is output repeatedly. A multiple word *expletive* must be enclosed in quotes. Termination is by the interrupt character (by default, CONTROL-c in the A/UX standard distribution).

FILES

/usr/ucb/yes

NAME

ypcat — list the contents of the named YP map

SYNOPSIS

ypcat [-k] [-t] [-d *domainname*] *mname*

ypcat -x

DESCRIPTION

ypcat lists the contents of the yellow pages (YP) map specified by *mname*, which may be either a *mapname* or a map "nickname." Since ypcat uses the YP network services, no YP server is specified.

Suppose you want to look at the system password file. Each machine has a file /etc/passwd, containing only a small amount of information. The network-wide password database is served by the YP, and holds information in a map named passwd.byname, whose abbreviated name is passwd. Thus, to see the system passwords, you type:

```
ypcat passwd
```

Here is a list of abbreviated mapnames, full mapnames, and corresponding system files:

passwd	pwd.nm	/etc/passwd
group	grp.nm	/etc/group
hosts	hst.ad	/etc/hosts
networks	ntw.ad	/etc/networks
services	svc.nm	/etc/services
protocols	ptc.nr	/etc/protocols
netgroup	netg	/etc/netgroup

Note: This implementation uses shorter abbreviated names than other implementations due to an 8 character filename limitation.

Refer to ypfiles(4) and ypserv(1M) for an overview of the yellow pages.

FLAG OPTIONS

The following flag options are interpreted by ypcat:

-k Display the keys for those maps in which the values are null or the key is not part of the value. (None of the maps derived from files that have an ASCII version in /etc fall into this class.)

-t Inhibit translation of *mname* to *mapname*. For example, `ypcat -t passwd` will fail because there is no map named `passwd`, whereas `ypcat passwd` will be translated to `ypcat passwd.byname`.

-d Specify a domain other that the default domain. The default domain is returned by *domainname*.

-x Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

FILES

`/usr/bin/ypcat`

SEE ALSO

`domainname`(1), `ypmatch`(1), `ypserv`(1M), `ypfiles`(4).
"Installing the Yellow Pages" in *A/UX Network System Administration.*

NAME

ypmatch — list the value of keys in a YP map

SYNOPSIS

ypmatch [-d*domain*] [-k] [-t] *key... mname*

ypmatch -x

DESCRIPTION

ypmatch lists the values associated with one or more keys from the yellow pages (YP) map (database) specified by a *mname*, which may be either a mapname or a map nickname.

Multiple keys can be specified; the same map will be searched for all . The keys must be exact values insofar as capitalization and length are concerned. No pattern matching is available. If a key is not matched, a diagnostic message is produced.

FLAG OPTIONS

The following flag options are interpreted by ypmatch:

-d Specify a domain other than the default domain.

-k Before displaying the value of a key, display the key itself, followed by a colon (:). This is useful only if the keys are not duplicated in the values, or you've specified so many keys that the output could be confusing.

-t Inhibit translation of nickname to mapname. For example,

```
ypmatch -t zippy passwd
```

will fail because there is no map named passwd, while

```
ypmatch zippy passwd
```

will be translated to

```
ypmatch zippy passwd.byname
```

-x Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

FILES

/usr/bin/ypmatch

SEE ALSO

ypcat(1), ypfiles(4).
A/UX Network System Administration.

NAME

yppasswd — change login password in yellow pages

SYNOPSIS

yppasswd [*name*]

DESCRIPTION

yppasswd changes (or installs) a password associated with the user *name* (your own name by default) in the yellow pages. The yellow pages password may be different from the one on your own machine.

yppasswd prompts for the old yellow pages password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the superuser may change a password; in either case you must prove you know the old password.

FILES

/usr/bin/yppasswd

SEE ALSO

passwd(1), yppasswdd(1M), ypfiles(4).
A/UX Network System Administration.

BUGS

The update protocol passes all the information to the server in one rpc call, without ever looking at it. Thus, if you type in your old password incorrectly, you will not be notified until after you have entered and verified your new password.

NAME

ypwhich — which host is the YP server or map master?

SYNOPSIS

ypwhich [-d[*domain*]] [-V1] [*hostname*]

ypwhich [-d[*domain*]] [-V2] [*hostname*]

ypwhich [-t *mapname*] [-d *domain*] -m [*mname*]

ypwhich -x

DESCRIPTION

ypwhich tells which YP server supplies yellow pages services to a YP client, or which is the master for a map. If invoked without arguments, it gives the YP server for the local machine. If *hostname* is specified, that machine is queried to find out which YP master it is using.

Refer to ypfiles(4) and ypserv(1M) for an overview of the yellow pages.

FLAG OPTIONS

The following flag options are interpreted by ypwhich:

-d [*domain*]	Use *domain* instead of the default domain.
-V1	Which server is serving version 1 YP protocol-speaking client processes?
-V2	Which server is serving version 2 YP protocol client processes?
	If neither version is specified, ypwhich attempts attempts to locate the server that supplies the (current) version 2 services. If there is no version 2 server currently bound, ypwhich then attempts to locate the server supplying the version 1 services. Since YP servers and YP clients are both backward compatible, the user need seldom be concerned about which version is currently in use.
-t *mapname*	Inhibit nickname translation; useful if there is a mapname identical to a nickname.
-m [*mname*]	Find the master YP server for a map. No *hostname* can be specified with -m. *mname* can be a mapname, or a nickname for a map.

`-x` Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

FILES

`/usr/bin/ypwhich`

SEE ALSO

`rpcinfo`(1M), `ypset`(1M), `ypserv`(1M), `ypfiles`(4).
A/UX Network System Administration.

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 directory /usr/games.

NAME

aliens — alien invaders attack the earth

SYNOPSIS

/usr/games/aliens

DESCRIPTION

This is an A/UX version of Space Invaders. The program is pretty much self-documenting.

FILES

/usr/games/aliens
/usr/games/alienslog

BUGS

The program is a CPU hog. It needs to be rewritten. It doesn't do well on terminals that run slower than 9600 baud.

NAME

arithmetic — provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [+-x/] [*range*]

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 first optional argument determines the kind of problem to be generated; +, -, x, and / respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is +-.

range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

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 *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

FILES

/usr/games/arithmetic

NAME

autorobots — escape from the automatic robots

SYNOPSIS

/usr/games/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 ten scores and prints them at the end of the game.

The valid commands are described on the screen.

FILES

/usr/games/autorobots

NAME

back — the game of backgammon

SYNOPSIS

/usr/games/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.

FILES

/usr/games/back
/usr/games/lib/backrules
$HOME/.backlog

BUGS

The only level really worth playing is expert, and it only plays the forward game.

Doubling is not implemented.

NAME

bcd — simulate a punched card corresponding to a text argument

SYNOPSIS

/usr/games/bcd *text*

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

NAME

bj — the game of black jack

SYNOPSIS

/usr/games/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.

A player 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, the player is 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 the player wins $2 if the dealer has a natural and loses $1 if the dealer does not.

If the player is dealt two cards of the same value, he is allowed to double. He is 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, the player may doubledown. He may double the bet ($2 to $4) and receive exactly one more card on that hand.

Under normal play, the player may hit (draw a card) as long as his total is not over twenty-one. If the player busts (goes over twenty-one), the dealer wins the bet.

When the player stands (decides not to hit), the dealer hits until he attains a total of seventeen or more. If the dealer busts, the player wins the bet.

If both player and 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`

NAME

chase — try to escape the killer robots

SYNOPSIS

/usr/games/chase [*nrobots*] [*nfences*]

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

NAME

craps — the game of craps

SYNOPSIS

/usr/games/craps

DESCRIPTION

craps is a form of the game of craps that is played in Las Vegas. The program simulates the *roller*, while the user (the *player*) places bets. The player 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.

The player starts off with a bankroll of $2,000.

The program prompts with:

```
bet?
```

The bet can be all or part of the player's 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 a player loses the entire bankroll, the House will offer to lend the player an additional $2,000. The program will prompt:

```
marker?
```

A `yes` (or `y`) consummates the loan. Any other reply terminates the game.

If a player owes the House money, the House reminds the player, before a bet is placed, how many markers are outstanding.

If, at any time, the bankroll of a player who has outstanding markers exceeds $2,000, the House asks:

```
Repay marker?
```

A reply of `yes` (or `y`) indicates the player's willingness to repay the loan. If only 1 marker is outstanding, it is immediately repaid. However, if more than 1 marker are outstanding, the House asks:

```
How many?
```

markers the player 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 a player accumulates 10 markers (a total of $20,000 borrowed from the House), the program informs the player of the situation and exits.

Should the bankroll of a player who has outstanding markers exceed $50,000, the *total* amount of money borrowed will be *automatically* repaid to the House.

Any player who accumulates $100,000 or more breaks 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 the player won, lost, or broke even.

MISCELLANEOUS

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`

NAME

cribbage — the card game cribbage

SYNOPSIS

/usr/games/cribbage [-r] [-e] [-q] *name...*

DESCRIPTION

cribbage plays the card game cribbage, with the program playing one hand and the user the other. The program will initially ask the user if the rules of the game are needed; if so, it will print out the appropriate section from *According to Hoyle* with more(1).

cribbage options include:

-e When the player makes a mistake scoring his hand or crib, provide an explanation of the correct score. (This is especially useful for beginning players.)

-q Print a shorter form of all messages; this is only recommended for users who have played the game without specifying this option.

-r Instead of asking the player to cut the deck, the program will randomly cut the deck.

cribbage first asks the player whether he wishes 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 the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who 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 the player's hand, whose crib it is, and then asks the player 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 the player's crib) or asks the player to cut the deck (if it's its crib); in the later 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 the player to score his hand (and the crib, if it is his) 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`. Suits may be specified as: `s`, `h`, `d`, and `c`, or alternatively as: `spades`, `hearts`, `diamonds`, and `clubs`. 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`

cubic(6)

See ttt(6)

NAME

fish — play "Go Fish"

SYNOPSIS

/usr/games/fish

DESCRIPTION

fish plays the game of "Go Fish," a childrens' card game. 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, he gives them 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 he had last requested, he draws 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

NAME

fortune — print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune

DESCRIPTION

fortune prints out a random adage.

FILES

/usr/games/fortune
/usr/games/lib/fortunes

NAME

hangman — guess the word

SYNOPSIS

/usr/games/hangman [*arg*]

DESCRIPTION

hangman chooses a word at least seven letters long from a dictionary. The user is to guess letters one at a time.

The optional argument *arg* names an alternate dictionary.

FILES

/usr/games/hangman
/usr/lib/w2006

BUGS

Hyphenated compounds are run together.

NAME

life — play the game of life

SYNOPSIS

/usr/games/life [-r]

DESCRIPTION

life is a pattern-generating game set up for interactive use on a video terminal. The way it operates is: You use a series of commands to set up a pattern on the screen and then let it generate further patterns from that pattern.

The algorithm used is this: For each square in the matrix, look at it and its eight adjacent neighbors. If the present square is unoccupied and exactly three of its neighboring squares are occupied, then that square will be occupied in the next pattern. If the present square is occupied and two or three 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. If you specify the -r option on the command line, the screen is treated as a torus; that is, the top and bottom lines and the left and right columns are considered adjacent.

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 — Add 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 — Step through the next *n* patterns. If no number is specified, step forever. The operation can be cancelled by typing an interrupt.

m,*n*d — Delete 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 — Generate a little flier at the present location. The number (modulo 8) determines the direction.

m,*n*g Move 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 Move left *n* steps. If no number is specified, the default is 1.

*n*j Move down *n* steps. The default is 1.

*n*k Move up *n* steps. The default is 1.

*n*l Move right *n* steps. The default is 1.

*n*n Step through the next *n* patterns. If no number is specified, generate the next pattern. The operation can be cancelled by typing an interrupt.

p Put the last yanked or deleted block at the present location.

q Quit.

m,*n*y Yank 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 Clear the pattern.

*n*F Generate a big flier at the present location. The number (modulo 8) determines the direction.

*n*H Move to the left margin.

*n*J Move to the bottom margin.

*n*K Move to the top margin.

*n*L Move to the right margin.

*n*CONTROL-H
Move left *n* steps. If no number is specified, the default is 1.

*n*CONTROL-J
Move down *n* steps. The default is 1.

*n*CONTROL-K
Move up *n* steps. The default is 1.

*n*CONTROL-L
Move right *n* steps. The default is 1.

CONTROL-R
: Redraw the screen. This is used for those occasions when the terminal screws up.

`.`
: Repeat the last add (`a`) or delete (`d`) operation.

`;`
: Repeat the last move (`h`, `j`, `k`, `l`) operation.

FILES

`/usr/games/life`

BUGS

The following features are planned but not implemented.

m,*n*`S`
: Save the selected area in a file.

`R`
: Restore from a file.

`m`
: Generate a macro command.

`!`
: Shell escape.

`e`
: Edit a file.

`i`
: Input commands from a file.

NAME

mastermind — play the game of Mastermind

SYNOPSIS

/usr/games/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	lock file
/usr/games/mmhow	instruction file

NAME

maze — generate a maze

SYNOPSIS

/usr/games/maze

DESCRIPTION

maze asks a few questions and then prints a maze.

FILES

/usr/games/maze

BUGS

Some mazes (especially small ones) have no solutions.

NAME

moo — guessing game

SYNOPSIS

`/usr/games/moo`

DESCRIPTION

`moo` is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. The player guesses 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 the player guesses the number (a score of four bulls).

FILES

`/usr/games/moo`

NAME

number — convert Arabic numerals to English

SYNOPSIS

/usr/games/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

NAME

quiz — test your knowledge

SYNOPSIS

/usr/games/quiz [-i*file*] [-t] [*category1 category2*]

DESCRIPTION

quiz gives associative knowledge tests on various subjects. It asks 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.

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.

The -t flag specifies tutorial mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The -i flag 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 file 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. Backslash \ is used as with sh(1) 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.

FILES

/usr/games/quiz
/usr/games/lib/quiz/index
/usr/games/lib/quiz/*

BUGS

The construct

```
a | ab
```

doesn't work in an information file. Use a{b}.

NAME

rain — animated raindrops display

SYNOPSIS

/usr/games/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
/etc/termcap

NAME

robots — escape from the robots

SYNOPSIS

/usr/games/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 ten scores and prints them at the end of the game.

The valid commands are described on the screen.

FILES

/usr/games/robots

NAME

trek — trekkie game

SYNOPSIS

/usr/games/trek [[-a] *file*]

DESCRIPTION

trek is a game of space glory and war. Below is a summary of commands. For complete documentation, see *Trek* by Eric Allman.

If a filename is given, a log of the game is written onto that file. If the -a flag is given before the filename, that file is appended to, not truncated.

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 a 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.

COMMAND SUMMARY

abandon
cloak up/down
computer request; ...
destruct
help
lrscan
phasers automatic amount
phasers manual amt1 course1 spread1 ...
torpedo course [yes] angle/no
ram course distance
shell
srscan [yes/no]
status
undock
warp warp_factor

capture
damages
dock
impulse course distance
move course distance
rest time
shields up/down
terminate yes/no
visual course

FILES

/usr/games/trek

NAME

ttt, cubic — tic-tac-toe

SYNOPSIS

/usr/games/ttt

/usr/games/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
/usr/games/ttt.k
/usr/games/cubic

NAME

`twinkle` — twinkle stars on the screen

SYNOPSIS

`/usr/games/twinkle` [-] [+] [s *save*]] [*density1* [*density2*]]

DESCRIPTION

`twinkle` causes a specified density of "stars" to twinkle on the screen.

FLAG OPTIONS

The following flag options are available;

- `-` Print 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.
- `+` Do 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.
- `s` Save binary density on file *save*, in case you want to see the density curve that a particular density specification produced during the life of the show.

density

If *density* 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*).

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`

NAME

worm — play the growing worm game

SYNOPSIS

/usr/games/worm [*size*]

DESCRIPTION

In worm, you are a little worm, your body is the os on the screen, and your head is the @, You move with the keys 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 several (9 for HL and 5 for JK) of the corresponding lowercase key (unless you run into a digit, then it stops).

On the screen you will see a digit; if your 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.

The optional argument, if present, is the initial length of the worm.

FILES

/usr/games/worm

BUGS

If the initial length of the worm is set to less than one or more than 75, various strange things happen.

NAME

worms — animate worms on a display terminal

SYNOPSIS

/usr/games/worms [-field] [-length *n*] [-number *n*] [-trail]

DESCRIPTION

-field makes a field for the worm(s) to eat; -trail causes each worm to leave a trail behind it. The rest is easily understood.

FILES

/usr/games/worms
/etc/termcap

DIAGNOSTICS

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.

Out of memory
: This should never happen.

BUGS

The lower right character position will not be properly updated on a terminal that wraps at the right margin.

Terminal initialization is not performed.

NAME

wump — the game of hunt-the-wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

wump plays the game of "Hunt the Wumpus". A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding 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 in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in *People's Computer Company*, 2, 2 (November 1973).

FILES

/usr/games/wump

BUGS

It will never replace Adventure.

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh® computers and troff running on A/UX. Proof and final pages were created on Apple LaserWriter® printers. POSTSCRIPT®, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

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

Writers: J. Eric Akin, Mike Elola, George Towner, and Kathy Wallace

Editor: George Truett

Production Supervisor: Josephine Manuele

Acknowledgments: Lori Falls and Michael Hinkson

Special thanks to Lorraine Aochi, Vicki Brown, Sharon Everson, Pete Ferrante, Kristi Fredrickson, Don Gentner, Tim Monroe, Dave Payne, Henry Seltzer, and John Sovereign

030-0782