UNIPLUS+ SYSTEM V

User's Manual

Section 1

UniSoft SYSTEMS

INTRODUCTION

This manual describes the features of System V UniPlus+, a UNIX operating system. All commands, features, and facilities described in this manual are available on UniPlus+.

This manual is divided into two volumes containing a total of six sections, some containing subsections:

1. Commands and Application Programs:
   1. General-Purpose Commands.
   1C. Communications Commands.
   1G. Graphics Commands.
2. System Calls.
3. Subroutines:
   3C. C and Assembler Library Routines.
   3M. Mathematical Library Routines.
   3S. Standard I/O Library Routines.
   3X. Miscellaneous Routines.
4. File Formats.
5. Miscellaneous Facilities.
6. Games.

Section 1 (*Commands and Application Programs*) describes programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory **/bin** (for **bin**ary programs). Some programs also reside in **/usr/bin**, to save space in **/bin**. These directories are searched automatically by the command interpreter called the *shell*. Sub-class 1C contains communication programs such as *cu*, *send*, *uucp*, etc.

Section 2 (*System Calls*) describes the entries into the UNIX kernel, including the C language interface.

Section 3 (*Subroutines*) describes the available subroutines. Their binary versions reside in various system libraries in the directories **/lib** and **/usr/lib**. See *intro*(3) for descriptions of these libraries and the files in which they are stored.

Section 4 (*File Formats*) documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in *a.out*(4). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language **struct** declarations corresponding to these formats can be found in the directories **/usr/include** and **/usr/include/sys**.

Section 5 (*Miscellaneous Facilities*) includes descriptions of character sets, macro packages and other system features.

Section 6 (*Games*) describes the games and educational programs that, as a rule, reside in the directory **/usr/games**.

Each section consists of a number of independent entries of a page or so each. The name of the entry appears in the upper corners of its pages. Entries within each

section are alphabetized, with the exception of the introductory entry that begins each section. The page numbers of each entry start at 1. The version date of the entry appears in the lower left corner of each page. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "major" name.

All entries are based on a common format, not all of whose parts always appear:

The **NAME** part gives the name(s) of the entry and briefly states its purpose.

The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (*Commands*):

Boldface strings are literals and are to be typed just as they appear.

Italic strings usually represent substitutable argument prototypes and program names found elsewhere in the manual.

Square brackets [] around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file", it always refers to a *file* name.

Ellipses ... are used to show that the previous argument prototype may be repeated.

A final convention is used by the commands themselves. An argument beginning with a minus −, plus +, or equal sign = is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with −, +, or =.

The **DESCRIPTION** part discusses the subject at hand.

The **EXAMPLE** part gives example(s) of usage, where appropriate.

The **FILES** part gives the file names that are built into the program.

The **SEE ALSO** part gives pointers to related information.

The **DIAGNOSTICS** part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The **WARNINGS** part points out potential pitfalls.

The **BUGS** part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

At the front of each volume there is a table of contents and a permuted index. The permuted index is a computer-generated index that uses the information in the **NAME** part of each entry in the User's and Administrator's Manuals. The permuted index contains three columns. The center column is an alphabetic list of keywords as they appear in the **NAME** part of the entries. The last column is the entry that the keyword in the center column refers to. This entry is followed by the appropriate section number in parentheses. The first column contains the remaining information from the **NAME** part that either precedes or follows the keyword.

For example, to look for a text editor, scan the center column for the word "editor". There are several index lines containing an "editor" reference, i.e.:

```
ed, red: text   editor.                    . . . . . ed(1)
files. ld: link editor for common object . . . . . ld(1)
```

You can then turn to the entries listed in the last column, *ed*(1) and *ld*(1), to find information on the editor.

On most systems, all user manual entries are available on-line via the command, q.v.

TABLE OF CONTENTS

1. Commands and Application Programs

PERMUTED INDEX

NAME

intro − introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, publicly-accessible commands. Certain distinctions of purpose are made in the headings:

(1) Commands of general utility.
(1C) Commands for communication with other systems.
(1G) Commands used primarily for graphics and computer-aided design.

COMMAND SYNTAX

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option*(*s*)] [*cmdarg*(*s*)]

where:

name	The name of an executable file.
option	− *noargletter* (*s*) or, − *argletter* < > *optarg* where < > is optional white space.
noargletter	A single letter representing an option without an argument.
argletter	A single letter representing an option requiring an argument.
optarg	Argument (character string) satisfying preceding *argletter*.
cmdarg	Path name (or other command argument) *not* beginning with − or, − by itself indicating the standard input.

SEE ALSO

getopt(1), getopt(3C).

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (see *wait*(2) and *exit*(2)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

BUGS

Regretfully, many commands do not adhere to the aforementioned syntax.

NAME

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

SYNOPSIS

300 [**+12**] [**−n**] [**−d**t,l,c]

300s [**+12**] [**−n**] [**−d**t,l,c]

DESCRIPTION

300 supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal; *300s* performs the same functions for the DASI 300s (GSI 300s or DTC 300s) terminal. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12-pitch text. It also reduces printing time 5 to 70%. *300* can be used to print equations neatly, in the sequence:

neqn file ... | nroff | 300

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

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

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

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

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

The value **−d0,1** is a good one to use for C programs that have many levels of indentation.

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

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

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

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

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

The *neqn* names of, and resulting output for, the Greek and special characters supported by *300* are shown in *greek*(5).

SEE ALSO

450(1), eqn(1), mesg(1), nroff(1), stty(1), tabs(1), tbl(1), tplot(1G), greek(5).

BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.
If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

NAME

4014 − paginator for the Tektronix 4014 terminal

SYNOPSIS

4014 [**−t**] [**−n**] [**−c**N] [**−p**L] [file]

DESCRIPTION

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

The command line options are:

−t Don't wait between pages (useful for directing output into a file).

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

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

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

SEE ALSO

pr(1), tc(1), troff(1).

NAME

450 − handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

450 supports special functions of, and optimizes the use of, the DASI 450 terminal, or any terminal that is functionally identical, such as the DIABLO 1620 or XEROX 1700. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols in the same manner as *300*(1). *450* can be used to print equations neatly, in the sequence:

neqn file ... | nroff | 450

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

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

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

nroff −T450 files ...

or

nroff −T450−12 files ...

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

The *neqn* names of, and resulting output for, the Greek and special characters supported by *450* are shown in *greek*(5).

SEE ALSO

300(1), eqn(1), mesg(1), nroff(1), stty(1), tabs(1), tbl(1), tplot(1G), greek(5).

BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.
If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

NAME

acctcom − search and print process accounting file(s)

SYNOPSIS

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

DESCRIPTION

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

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

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

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

−b	Read backwards, showing latest commands first.
−f	Print the *fork/exec* flag and system exit status columns in the output.
−h	Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as: (total CPU time)/(elapsed time).
−i	Print columns containing the I/O counts in the output.
−k	Instead of memory size, show total kcore-minutes.
−m	Show mean core size (the default).
−r	Show CPU factor (user time/(system-time + user-time).
−t	Show separate system and user CPU times.
−v	Exclude column headings from the output.
−l *line*	Show only processes belonging to terminal **/dev/***line*.
−u *user*	Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a **#** which designates only those processes executed with *super-user* privileges, or **?** which designates only those processes associated with unknown user IDs.
−g *group*	Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.
−d *mm/dd*	Any *time* arguments following this flag are assumed to occur on the given month *mm* and the day *dd* rather than during last 24 hours. This is needed for looking at old files.
−s *time*	Select processes existing at or after *time*, given in the format *hr*[:*min*[:*sec*]].
−e *time*	Select processes existing at or before *time*.
−S *time*	Select processes starting at or after *time*.

−E *time* Select processes ending at or before *time*.

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

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

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

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

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

Listing options together has the effect of a logical *and.*

FILES

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

SEE ALSO

ps(1), su(1), acct(2), acct(4), utmp(4).
acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M) in the *UniPlus*$^{+}$ *Administrator's Manual.*

BUGS

Acctcom only reports on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time and option **−d** is not used, then *time* is interpreted as occurring on the previous day.

NAME

adb − debugger

SYNOPSIS

adb [−**w**] [−**k**] [objfil [corfil]]

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not, then the symbolic features of *adb* cannot be used although the file can still be examined. 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.**

Requests to *adb* are read from the standard input and responses are to the standard output. If the −**w** flag is present, then both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using *adb*. *Adb* ignores QUIT; INTERRUPT causes return to the next *adb* command.

To EXIT *adb*: use $q or $Q or Control-d.

Normally, for portability, *adb* does a system call to gather information regarding relocation addresses. If using *adb* on a stand-alone program, such as the kernel,**/unix**, use the −**k** flag which skips that part of the *adb* code.

In general requests to *adb* are of the form

[*address*] [, *count*] [*command*] [;]

If *address* is present, then *dot* is set to *address*. Initially *dot* is set to 0. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *Address* and *count* are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged, then addresses are interpreted in the usual way in the address space of the subprocess. If the operating system is being debugged either post-mortem or using the special file **/dev/kmem** to interactive examine and/or modify memory, the maps are set to map the kernel virtual addresses. For further details of address mapping see ADDRESSES.

EXPRESSIONS

.	The value of *dot*.
+	The value of *dot* incremented by the current increment.
^	The value of *dot* decremented by the current increment.
"	The last *address* typed.
integer	A number. The prefix 0 (zero) forces interpretation in octal radix; the prefixes 0d and 0D force interpretation in decimal radix; the prefixes 0x and 0X force interpretation in hexadecimal radix. Thus 020 = 0d16 = 0x10 = sixteen. If no prefix appears, then the *default radix* is used; see the $d command. The default radix is initially hexadecimal. The hexadecimal digits are 0123456789abcdefABCDEF with the obvious values. Note that a

hexadecimal number whose most significant digit would otherwise be an alphabetic character must have a 0x (or 0X) prefix (or a leading zero if the default radix is hexadecimal).

integer.fraction
A 32-bit floating point number.

'cccc' The ASCII value of up to 4 characters. \ may be used to escape a '.

< name The value of *name*, which is either a variable name or a register name. *Adb* maintains a number of variables (see VARIABLES) named by single letters or digits. If *name* is a register name, then the value of the register is obtained from the system header in *corfil*. The register names are those printed by the $r command.

symbol A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. \ may be used to escape other characters. The value of the *symbol* is taken from the symbol table in *objfil*. An initial _ or ~ will be prepended to *symbol* if needed.

_ symbol In C, the "true name" of an external symbol begins with _ . It may be necessary to utter this name to distinguish it from internal or hidden variables of a program.

(exp) The value of the expression *exp*.

Monadic operators:

**exp* The contents of the location addressed by *exp* in *corfil*.

The contents of the location addressed by *exp* in *objfil*.

−exp Integer negation.

~exp Bitwise complement.

#exp Logical negation.

Dyadic operators are left associative and are less binding than monadic operators.

e1+e2 Integer addition.

e1−e2 Integer subtraction.

*e1*e2* Integer multiplication.

e1%e2 Integer division.

e1&e2 Bitwise conjunction.

e1|e2 Bitwise disjunction.

e1#e2 *E1* rounded up to the next multiple of *e2*.

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands "?" and "/" may be followed by "*"; see ADDRESSES for further details.)

?f Locations starting at *address* in *objfil* are printed according to the format *f*. *Dot* is incremented by the sum of the increments for each format letter (q.v.).

/f Locations starting at *address* in *corfil* are printed according to the format *f*, and *dot* is incremented as for "?".

=f The value of *address* itself is printed in the styles indicated by the format *f*. (For **i** format "?" is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format, *dot* is incremented by the amount given for each format letter. If no format is given, then the last format is used. The format letters available are as follows.

i	*n*	Disassemble the addressed instruction.
o	2	Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
O	4	Print 4 bytes in octal.
q	2	Print in signed octal.
Q	4	Print long signed octal.
d	2	Print in decimal.
D	4	Print long decimal.
x	2	Print 2 bytes in hexadecimal.
X	4	Print 4 bytes in hexadecimal.
u	2	Print as an unsigned decimal number.
U	4	Print long unsigned decimal.
f	4	Print the 32-bit value as a floating point number.
F	8	Print double floating point.
b	1	Print the addressed byte in octal.
c	1	Print the addressed character.
C	1	Print the addressed character using the standard escape convention where control characters are printed as ^*X* and the delete character is printed as ^*?*.
s	*n*	Print the addressed characters until a zero character is reached.
S	*n*	Print a string using the ^*X* escape convention (see **C** above). The *n* is the length of the string including its zero terminator.
Y	4	Print 4 bytes in date format (see *ctime*(3)).
a	0	Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below. / global data symbol ? global text symbol = global absolute symbol
p	4	Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.
t	0	When preceded by an integer tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.
r	0	Print a space.
n	0	Print a newline.
"..."	0	Print the enclosed string.
^		*Dot* is decremented by the current increment. Nothing is printed.
+		*Dot* is incremented by 1. Nothing is printed.
−		*Dot* is decremented by 1. Nothing is printed.

newline
Repeat the previous command with a *count* of 1.

[?/]l *value mask*
Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If L is used, then the match is for 4 bytes at a time instead of 2. If no match is found, then *dot* is unchanged; otherwise, *dot* is set to the matched location. If *mask* is omitted, then −1 is used.

[?/]w *value ...*
Write the 2-byte *value* into the addressed location. If the command is W, write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m *b1 e1 f1*[?/]
New values for (*b1*, e1, *f1*) are recorded. If less than three expressions are given, then the remaining map parameters are left unchanged. If the "?" or "/" is followed by "*", then the second segment (*b2*, *e2*, *f2*) of the mapping is changed. If the list is terminated by "?" or "/", then the file (*objfil* or *corfil* respectively) is used for subsequent requests. (So that, for example, "/m?" will cause "/" to refer to *objfil*.)

>*name*
Dot is assigned to the variable or register named.

! A shell is called to read the rest of the line following "!".

$*modifier*
Miscellaneous commands. The available *modifiers* are:

- <*f* Read commands from the file *f*. If this command is executed in a file, further commands in the file are not seen. If *f* is omitted, the current input stream is terminated. If a *count* is given, and is zero, the command will be ignored. The value of the count will be placed in variable *9* before the first command in *f* is executed.
- <<*f* Similar to < except it can be used in a file of commands without causing the file to be closed. Variable *9* is saved during the execution of this command, and restored when it completes. There is a (small) finite limit to the number of << files that can be open at once.
- >*f* Append output to the file *f*, which is created if it does not exist. If *f* is omitted, output is returned to the terminal.
- ? Print process ID, the signal which caused stoppage or termination, as well as the registers as **$r**. This is the default if *modifier* is omitted.
- **r** Print the general registers and the instruction addressed by **pc**. *Dot* is set to **pc**.
- **b** Print all breakpoints and their associated counts and commands.
- **c** C stack backtrace. If *address* is given, then it is taken as the address of the current frame (instead of **a7**). If **C** is used, then the names and (16 bit) values of all automatic and static variables are printed for each active function. If *count* is given, then only the first *count* frames are printed.

d Set the default radix to *address* and report the new value. Note that *address* is interpreted in the (old) current radix. Thus 10$d never changes the default radix. To make decimal the default radix, use 0t10$d.
e The names and values of external variables are printed.
w Set the page width for output to *address* (default 80).
s Set the limit for symbol matches to *address* (default 255).
o All integers input are regarded as octal.
d Reset integer input as described in EXPRESSIONS.
q Exit from *adb.*
v Print all non zero variables in octal.
m Print the address map.

:*modifier*

Manage a subprocess. Available modifiers are:

b*c* Set breakpoint at *address*. The breakpoint is executed *count*−1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command is omitted or sets *dot* to zero then the breakpoint causes a stop.
d Delete breakpoint at .IR address .
r Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the subprocess.
c*s* The subprocess is continued with signal *s* (see *signal*(2)). If *address* is given, then the subprocess is continued at this address. If no signal is specified, then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.
s*s* As for **c** except that the subprocess is single stepped *count* times. If there is no current subprocess then *objfil* is run as a subprocess as for **r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.
k The current subprocess, if any, is terminated.

VARIABLES

Adb provides a number of variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows.

0 The last value printed.
1 The last offset part of an instruction source.
2 The previous value of variable 1.
9 The count on the last $< or $<< command.

On entry the following are set from the system header in the *corfil.* If *corfil* does not appear to be a **core** file, then these values are set from *objfil.*

b The base address of the data segment.
d The data segment size.
e The entry point.

m The "magic" number (0407, 0410).
s The stack segment size.
t The text segment size.

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* and the *file address* corresponding to a written *address* is calculated as follows.

$b1 \leqslant address < e1 \Rightarrow$ *file address* $= address + f1 - b1$, otherwise,

$b2 \leqslant address < e2 \Rightarrow$ *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. If a ? or / is followed by an *, then only the second triple is used.

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.

So that *adb* may be used on large files all appropriate values are kept as signed 32-bit integers.

EXAMPLE

```
adb obj1
```

will invoke *adb* with the executable object "obj1"; when *adb* responds with

```
ready
```

the request:

```
main,10?ia
```

will cause 16 (10hex) instructions to be printed in assembly code, starting from location "main".

FILES

a.out
core

SEE ALSO

a.out(4), core(4)

DIAGNOSTICS

Adb when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned nonzero status.

BUGS

Use of # for the unary logical negation operator is peculiar.

There doesn't seem to be any way to clear all breakpoints.

NAME

admin − create and administer SCCS files

SYNOPSIS

admin [−**n**] [−**i**[name]] [−**r**rel] [−**t**[name]] [−**f**flag[flag-val]] [−**d**flag[flag-val]] [−**a**login] [−**e**login] [−**m**[mrlist]] [−**y**[comment]] [−**h**] [−**z**] files

DESCRIPTION

Admin is used to create new SCCS files and change parameters of existing ones. Arguments to *admin*, which may appear in any order, consist of keyletter arguments, which begin with −, and named files (note that SCCS file names must begin with the characters **s.**). If a named file doesn't exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

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

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

−n This keyletter indicates that a new SCCS file is to be created.

−i[*name*] The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see **−r** keyletter for delta numbering scheme). If the **i** keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an *admin* command on which the **i** keyletter is supplied. Using a single *admin* to create two or more SCCS files require that they be created empty (no **−i** keyletter). Note that the **−i** keyletter implies the **−n** keyletter.

−r*rel* The *rel*ease into which the initial delta is inserted. This keyletter may be used only if the **−i** keyletter is also used. If the **−r** keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

−t[*name*] The *name* of a file from which descriptive text for the SCCS file is to be taken. If the **−t** keyletter is used and *admin* is creating a new SCCS file (the **−n** and/or **−i** keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a **−t** keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a **−t** keyletter with a file name causes text (if any) in the named file to replace the

descriptive text (if any) currently in the SCCS file.

−f*flag* This keyletter specifies a *flag*, and , possibly, a value for the *flag*, to be placed in the SCCS file. Several **f** keyletters may be supplied on a single *admin* command line. The allowable *flag*s and their values are:

b Allows use of the **−b** keyletter on a *get*(1) command to create branch deltas.

c*ceil* The highest release (i.e., "ceiling"), a number less than or equal to 9999, which may be retrieved by a *get*(1) command for editing. The default value for an unspecified **c** flag is 9999.

f*floor* The lowest release (i.e., "floor"), a number greater than 0 but less than 9999, which may be retrieved by a *get*(1) command for editing. The default value for an unspecified **f** flag is 1.

d*SID* The default delta number (SID) to be used by a *get*(1) command.

i Causes the "No id keywords (ge6)" message issued by *get*(1) or *delta*(1) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get*(1)) are found in the text retrieved or stored in the SCCS file.

j Allows concurrent *get*(1) commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

l*list* A *list* of releases to which deltas can no longer be made (**get −e** against one of these "locked" releases fails). The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range>::= *RELEASE NUMBER* | **a**

The character **a** in the *list* is equivalent to specifying *all releases* for the named SCCS file.

n Causes *delta*(1) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file preventing branch deltas from being created from them in the future.

q*text* User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by *get*(1).

m*mod* *Mod*ule name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text

retrieved by *get*(1). If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s.** removed.

t*type* *Type* of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by *get*(1).

v[*pgm*] Causes *delta*(1) to prompt for Modification Request (*MR*) numbers as the reason for creating a delta. The optional value specifies the name of an *MR* number validity checking program (see *delta*(1)). (If this flag is set when creating an SCCS file, the **m** keyletter must also be used even if its value is null).

−d*flag* Causes removal (deletion) of the specified *flag* from an SCCS file. The **−d** keyletter may be specified only when processing existing SCCS files. Several **−d** keyletters may be supplied on a single *admin* command. See the **−f** keyletter for allowable *flag* names.

l*list* A *list* of releases to be "unlocked". See the **−f** keyletter for a description of the **l** flag and the syntax of a *list*.

−a*login* A *login* name, or numerical UNIX System group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several **a** keyletters may be used on a single *admin* command line. As many *login*s, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas.

−e*login* A *login* name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several **e** keyletters may be used on a single *admin* command line.

−y[*comment*] The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*(1). Omission of the **−y** keyletter results in a default comment line being inserted in the form:

date and time created *YY/ MM/ DD HH:MM:SS* by *login*

The **−y** keyletter is valid only if the **−i** and/or **−n** keyletters are specified (i.e., a new SCCS file is being created).

−m[*mrlist*] The list of Modification Requests (*MR*) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*(1). The **v** flag must be set and the *MR* numbers are validated if the **v** flag has a value (the name of an *MR* number validation program). Diagnostics will occur if the **v** flag is not set or *MR* validation fails.

−h Causes *admin* to check the structure of the SCCS file (see *sccsfile*(5)), and to compare a newly computed check-sum

(the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

-z The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see **-h**, above).

Note that use of this keyletter on a truly corrupted file may prevent future detection of the corruption.

EXAMPLE

```
admin -i file1 s.file1
```

creates a new file in SCCS format named "s.file1", from "file1".

FILES

The last component of all SCCS file names must be of the form **s.***file-name*. New SCCS files are given mode 444 (see *chmod*(1)). Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file, called **x.***file-name*, (see *get*(1)), created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of *ed*(1). *Care must be taken!* The edited file should *always* be processed by an **admin -h** to check for corruption followed by an **admin -z** to generate a proper check-sum. Another **admin -h** is recommended to ensure the SCCS file is valid.

Admin also makes use of a transient lock file (called **z.***file-name*), which is used to prevent simultaneous updates to the SCCS file by different users. See *get*(1) for further information.

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(4).
Source Code Control System User's Guide

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ar — archive and library maintainer

SYNOPSIS

ar [uvnbail] [mrxtdpq] [posname] archivename filename(s) ...

DESCRIPTION

The archive command *ar* maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the loader. However, *ar* can be used for any similar archiving purpose. Archives often consist of unlinked program modules.

Key is *one* character from the set **mrxtdpq**, optionally concatenated with one or more of **uvnbail**. *Archivename* is the archive file. The *filename(s)* are constituent files in or destined for the archive file. The meanings of the *key* characters are:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.

t Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p Print the named files in the archive.

m Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.

v Verbose. Under the verbose option, *ar* gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of all information about the files. When used with **p**, it precedes each file with a name.

c Create. Normally *ar* will create *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.

l Local. Normally *ar* places its temporary files in the directory **/tmp**. This option causes them to be placed in the local directory.

EXAMPLE

```
ar rv libar.a text.o
```

places file "text.o" in archive "libar.a".

```
ar bm file1 archivename file2
```

changes the location of a file inside an archive. "File2" is the file to be moved. "File2" is moved to a new position before "file1".

FILES

/tmp temporaries

SEE ALSO

ld(1), ar(4).

BUGS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.
Sufficient disk space must be present to make an entire copy of the archive or the *ar* command will fail.

NAME

as − assembler

SYNOPSIS

as [−**o** objfile] [−**v**] [−**l**] [name ...]

DESCRIPTION

As assembles the named files, or the standard input if no file name is specified.

All undefined symbols in the assembly are treated as global.

The relocatable output of the assembly is left on the file *objfile*; if that is omitted, *a.out* is used.

The − **v** option enables **as** to recognize 68010 instruction mnemonics.

The −**l** option produces an assembly listing on file *objfile.lst*. If the −**l** option is specified and no −**o** parameter is specified, the assembly listing is placed on *a.lst*.

EXAMPLE

```
as −o file.o filea fileb filec
```

would assemble the three named files and put the output of the assembly into "file.o".

FILES

/tmp/as*	default temporary file
a.out	default resultant object file
a.lst	default assembly listing file

SEE ALSO

adb(1), ld(1), nm(1), a.out(4)
AS Assembler Reference Guide, James L. Gula and Thomas J. Teixeira. Revised by UniSoft Systems.

NAME

asa − interpret ASA carriage control characters

SYNOPSIS

asa [files]

DESCRIPTION

Asa interprets the output of FORTRAN programs that utilize ASA carriage control characters. It processes either the *files* whose names are given as arguments or the standard input if no file names are supplied. The first character of each line is assumed to be a control character; their meanings are:

' ' (blank) single new line before printing

0 double new line before printing

1 new page before printing

\+ overprint previous line.

Lines beginning with other than the above characters are treated as if they began with ' '. The first character of a line is *not* printed. If any such lines appear, an appropriate diagnostic will appear on standard error. This program forces the first line of each input file to start on a new page.

EXAMPLE

To correctly view the output of FORTRAN programs which use ASA carriage control characters, *asa* could be used as a filter thusly:

```
a.out | asa | lpr
```

and the output, properly formatted and paginated, would be directed to the line printer. FORTRAN output sent to a file could be viewed by:

```
asa file
```

SEE ALSO

efl(1), fortran(1), fsplit(1).

NAME

at − execute commands at a later time

SYNOPSIS

at time [day] [file]

DESCRIPTION

At squirrels away a copy of the named *file* (standard input default) to be used as input to *sh*(1) at a specified later time. A *cd*(1) command to the current directory is inserted at the beginning, followed by assignments to all environment variables. When the script is run, it uses the user and group ID of the creator of the copy file.

The *time* is 1 to 4 digits, with an optional following "A", "P", "N" or "M" for AM, PM, noon or midnight. One and two digit numbers are taken to be hours, three and four digits to be hours and minutes. If no letters follow the digits, a 24-hour clock time is understood.

The optional *day* is either (1) a month name followed by a day number, or (2) a day of the week; if the word "week" follows invocation is moved seven days further off. Names of months and days may be recognizably truncated. Examples of legitimate commands are

```
at 8am jan 24
at 1530 fr week
```

At programs are executed by periodic execution of the command */usr/lib/atrun* from *cron*(1M). The granularity of *at* depends upon how often *atrun* is executed.

Standard output or error output is lost unless redirected. The directory **/usr/spool/at/past** must be present or *at* will not run.

EXAMPLE

```
at 10:25
ls -l /etc > /dev/console
```

will cause the directory **/etc** to be listed in long format on device **/dev/console** at approximately 10:25 pm on the same day. The exact time this is executed will depend on how often */usr/lib/atrun* is scheduled to run in **/usr/lib/crontab**; e.g., if */usr/lib/atrun* is set up to run every 15 minutes, the above command will be executed at 10:30 am. A temporary file is created in directory **/usr/spool/at** containing the "ls -l" command to be executed; this temporary file will be removed upon completion of the command. Note that **/usr/lib/crontab** must contain a schedule entry for */usr/lib/atrun* in order for "at" to work.

FILES

/usr/spool/at/yy.ddd.hhhh.uu	activity to be performed at hour *hhhh* of day *ddd* of year *yy*. *uu* is a unique number.
/usr/spool/at/lasttimedone	contains *hhhh* for last hour of activity.
/usr/spool/at/past	directory of activities now in progress.
/usr/lib/atrun	program that executes activities that are due.
/usr/lib/crontab	cron table entry for running atrun.

SEE ALSO

calendar(1),
cron(1M) in the *UniPlus+ Administrator's Guide.*

DIAGNOSTICS

Complains about various syntax errors and times out of range.

BUGS

Due to the granularity of the execution of */usr/lib/atrun*, there may be bugs in scheduling things almost exactly 24 hours into the future.

NAME

awk − pattern scanning and processing language

SYNOPSIS

awk [−Fc] [prog] [parameters] [files]

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as **−f** *file*. The *prog* string should be enclosed in single quotes (') to protect it from the shell.

Parameters, in the form x=... y=... etc., may be passed to *awk*.

Files are read in order; if there are no files, the standard input is read. The file name − means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using FS, see below). The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

A pattern-action statement has the form:

pattern { action }

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next    # skip remaining patterns on this input line
exit    # skip the rest of the input
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the operators +, −, *, /, %, and concatenation (indicated by a blank). The C operators ++, −−, +=, −=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (").

The *print* statement prints its arguments on the standard output (or on a file if *>expr* is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its expression list according to the format (see *printf*(3S)).

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp*, *log*, *sqrt*, and *int*. The last truncates its argument to an integer; *substr* (*s*,*m*,*n* returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf* (*fmt*,*expr*,*expr* formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

Patterns are arbitrary Boolean combinations (!, ||, &&, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep* (see *grep*(1)). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression

where a relop is any of the six relational operators in C, and a matchop is either ~ (for *contains*) or !~ (for *does not contain*). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character *c* may be used to separate the fields by starting the program with:

BEGIN { FS = *c* }

or by using the **−F***c* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default new-line); and OFMT, the output format for numbers (default **%.6g**).

EXAMPLE

awk "length > 72" filea

prints lines longer than 72 characters on the standard output.

awk '{ print $2, $1 }' filea

prints the first two fields of each line in opposite order.

awk '{ s += $1 } END {print "sum is", s, "average is", s/NR }' filea

adds up the first column and prints the sum and average.

awk '{ for (i = NF; i > 0; −−i) print $i }' filea

prints all the fields of each line in reverse order. The output prints one field per line, beginning at the end of the file, unless otherwise directed.

awk "/start/, /stop/" filea

prints all lines between start/stop pattern pairs, for every such pair in the file.

SEE ALSO

grep(1), lex(1), sed(1).
Awk—A Pattern Scanning and Processing Language by A. V. Aho, B. W. Kernighan, and P. J. Weinberger.

BUGS

Input white space is not preserved on output if fields are involved.
There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

NAME

banner — make posters

SYNOPSIS

banner strings

DESCRIPTION

Banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

EXAMPLE

banner asa

will cause the characters "a", "s" and "a" to be printed as large letters on the screen.

SEE ALSO

echo(1).

NAME

banner7 − print large banner on printer

SYNOPSIS

banner7 [−w*n*] message ...

DESCRIPTION

Banner7 prints a large, high quality banner on the standard output. If the message is omitted, it prompts for and reads one line of its standard input. If −**w** is given, the output is scrunched down from a width of 132 to *n*, suitable for a narrow terminal. If *n* is omitted, it defaults to 80.

The output should be printed on a hard-copy device, up to 132 columns wide, with no breaks between the pages. The volume is enough that you want a printer or a fast hardcopy terminal, but if you are patient, a decwriter or other 300 baud terminal will do.

BUGS

Several ASCII characters are not defined, notably <, >, [,], \, ^, _, {, }, |, and ~. Also, the characters ", ', and & are funny looking (but in a useful way.)

The −**w** option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

AUTHOR

Mark Horton

NAME

basename, dirname − deliver portions of path names

SYNOPSIS

basename string [suffix]
dirname string

DESCRIPTION

Basename deletes any prefix ending in / and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside substitution marks (‘ ‘) within shell procedures.

Dirname delivers all but the last level of the path name in *string*.

EXAMPLE

Invoked with the argument **/usr/src/cmd/cat.c**,

```
cc $1
mv a.out ‘basename $1 .c‘
```

compiles the named file and moves the output to a file named "cat" in the current directory.

```
NAME=‘dirname /usr/src/cmd/cat.c‘
```

sets the shell variable NAME to **/usr/src/cmd**.

SEE ALSO

sh(1).

BUGS

The *basename* of / is null and is considered an error.

NAME

bc − arbitrary-precision arithmetic language

SYNOPSIS

bc [**−c**] [**−l**] [file ...]

DESCRIPTION

Bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The **−l** argument stands for the name of an arbitrary precision math library. The syntax for *bc* programs is as follows; L means letter a−z, E means expression, S means statement.

Comments
are enclosed in /* and */.

Names
simple variables: L
array elements: L [E]
The words "ibase", "obase", and "scale"

Other operands
arbitrarily long numbers with optional sign and decimal point.
(E)
sqrt (E)
length (E) number of significant decimal digits
scale (E) number of digits right of decimal point
L (E , ... , E)

Operators
+ − * / % ^ (% is remainder; ^ is power)
++ −− (prefix and postfix; apply to names)
== <= >= != < >
= =+ =− =* =/ =% =^

Statements
E
{ S ; ... ; S }
if (E) S
while (E) S
for (E ; E ; E) S
null statement
break
quit

Function definitions
define L (L ,..., L) {
auto L, ... , L
S; ... S
return (E)
}

Functions in −l math library
s(x) sine
c(x) cosine
e(x) exponential
l(x) log
a(x) arctangent
j(n,x) Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or new-lines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc*(1). Assignments to *ibase* or *obase* set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. "Auto" variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables empty square brackets must follow the array name.

Bc is actually a preprocessor for *dc*(1), which it invokes automatically, unless the **-c** (compile only) option is present. In this case the *dc* input is sent to the standard output instead.

EXAMPLE

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

defines a function to compute an approximate value of the exponential function and

```
for(i=1; i<=10; i++) e(i)
```

prints approximate values of the exponential function of the first ten integers.

FILES

/usr/lib/lib.b mathematical library
/usr/bin/dc desk calculator proper

SEE ALSO

dc(1).
BC—An Arbitrary Precision Desk-Calculator Language by L. L. Cherry and R. Morris.

BUGS

No **&&**, **||** yet.
For statement must have all three E's.
Quit is interpreted when read, not when executed.

NAME

bdiff − big diff

SYNOPSIS

bdiff file1 file2 [n] [−s]

DESCRIPTION

Bdiff is used in a manner analogous to *diff*(1) to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for *diff*. *Bdiff* ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. The value of *n* is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail. If *file1* (*file2*) is −, the standard input is read. The optional −s (silent) argument specifies that no diagnostics are to be printed by *bdiff* (note, however, that this does not suppress possible exclamations by *diff*. If both optional arguments are specified, they must appear in the order indicated above.

The output of *bdiff* is exactly that of *diff*, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

EXAMPLE

See example in *diff*(1).

FILES

/tmp/bd?????

SEE ALSO

diff(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

bfs – big file scanner

SYNOPSIS

bfs [–] name

DESCRIPTION

Bfs is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes (the maximum possible size) and 32K lines, with up to 255 characters per line. *Bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit*(1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the **w** command. The optional – suppresses printing of sizes. Input is prompted with * if **P** and a carriage return are typed as in *ed*. Prompting can be turned off again by inputting another **P** and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols besides / and ?: > indicates downward search without wrap-around, and < indicates upward search without wrap-around. Since *bfs* uses a different regular expression-matching routine from *ed*, the regular expressions accepted are slightly wider in scope (see *regcmp*(3X)). There is a slight difference in mark names: only the letters **a** through **z** may be used, and all 26 marks are remembered.

The **e**, **g**, **v**, **k**, **n**, **p**, **q**, **w**, =, ! and null commands operate as described under *ed*. Commands such as **– – –**, **+ + + –**, **+ + + =**, **–12**, and **+4p** are accepted. Note that **1,10p** and **1,10** will both print the first ten lines. The **f** command only prints the name of the file being scanned; there is no *remembered* file name. The **w** command is independent of output diversion, truncation, or crunching (see the **xo**, **xt** and **xc** commands, below). The following additional commands are available:

xf *file*
Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the **xf**. **Xf** commands may be nested to a depth of 10.

xo [*file*]
Further output from the **p** and null commands is diverted to the named *file*, which, if necessary, is created mode 666. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: *label*
This positions a *label* in a command file. The *label* is terminated by new-line, and blanks between the : and the start of the *label* are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(. , .)**xb**/ *regular expression*/ *label*
A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following

conditions:

1. Either address is not between **1** and **$**.
2. The second address is less than the first.
3. The regular expression doesn't match at least one line in the specified range, including the first and last lines.

On success, **.** is set to the line matched and a jump is made to *label*. This command is the only one that doesn't issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

```
xb/^/ label
```

is an unconditional jump.
The **xb** command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe only a downward jump is possible.

xt *number*
Output from the **p** and null commands is truncated to at most *number* characters. The initial number is 255.

xv[*digit*] [*spaces*] [*value*]
The variable name is the specified *digit* following the **xv**. **xv5100** or **xv5 100** both assign the value **100** to the variable **5**. **Xv61,100p** assigns the value **1,100p** to the variable **6**. To reference a variable, put a **%** in front of the variable name. For example, using the above assignments for variables **5** and **6**:

```
1,%5p
1,%5
%6
```

will all print the first 100 lines.

```
g/%5/p
```

would globally search for the characters **100** and print each line containing a match. To escape the special meaning of **%**, a **** must precede it.

```
g/".*\%[cds]/p
```

could be used to match and list lines containing *printf* of characters, decimal integers, or strings.
Another feature of the **xv** command is that the first line of output from a UNIX System command can be stored into a variable. The only requirement is that the first character of *value* be an **!**. For example:

```
.w junk
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

would put the current line into variable **5**, print it, and increment the variable **6** by one. To escape the special meaning of

! as the first character of *value*, precede it with a \.

```
xv7\!date
```

stores the value **!date** into variable **7**.

xbz *label*

xbn *label*

These two commands will test the last saved *return code* from the execution of a UNIX System command (*!command*) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string **size**.

```
xv55
: l
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn l
xv45
: l
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz l
```

xc [*switch*]

If *switch* is **1**, output from the **p** and null commands is crunched; if *switch* is **0** it isn't. Without an argument, **xc** reverses *switch*. Initially *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

EXAMPLE

```
bfs text
```

will invoke *bfs* with the file named "text".

SEE ALSO

csplit(1), ed(1), regcmp(3X).

DIAGNOSTICS

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

NAME

bs — a compiler/interpreter for modest-sized programs

SYNOPSIS

bs [file [args]]

DESCRIPTION

Bs is a remote descendant of Basic and Snobol4 with a little C language thrown in. *Bs* is designed for programming tasks where program development time is as important as the resulting speed of execution. Formalities of data declaration and file/process manipulation are minimized. Line-at-a-time debugging, the *trace* and *dump* statements, and useful run-time error messages all simplify program testing. Furthermore, incomplete programs can be debugged; *inner* functions can be tested before *outer* functions have been written and vice versa.

If the command line *file* argument is provided, the file is used for input before the console is read. By default, statements read from the file argument are compiled for later execution. Likewise, statements entered from the console are normally executed immediately (see *compile* and *execute* below). Unless the final operation is assignment, the result of an immediate expression statement is printed.

Bs programs are made up of input lines. If the last character on a line is a \, the line is continued. *Bs* accepts lines of the following form:

```
statement
label statement
```

A label is a *name* (see below) followed by a colon. A label and a variable can have the same name.

A *bs* statement is either an expression or a keyword followed by zero or more expressions. Some keywords (*clear*, *compile*, *!*, *execute*, *include*, *ibase*, *obase*, and *run*) are always executed as they are compiled.

Statement Syntax:

expression
: The expression is executed for its side effects (value, assignment or function call). The details of expressions follow the description of statement types below.

break
: *Break* exits from the inner-most *for/while* loop.

clear
: Clears the symbol table and compiled statements. *Clear* is executed immediately.

compile [expression]
: Succeeding statements are compiled (overrides the immediate execution default). The optional expression is evaluated and used as a file name for further input. A *clear* is associated with this latter case. *Compile* is executed immediately.

continue
: *Continue* transfers to the loop-continuation of the current *for/while* loop.

dump [name]
: The name and current value of every non-local variable is printed.

Optionally, only the named variable is reported. After an error or interrupt, the number of the last statement and (possibly) the user-function trace are displayed.

exit [expression]
Return to system level. The expression is returned as process status.

execute
Change to immediate execution mode (an interrupt has a similar effect). This statement does not cause stored statements to execute (see *run* below).

for name = expression expression statement
for name = expression expression
...
next

for expression , expression , expression statement
for expression , expression , expression
...
next
The *for* statement repetitively executes a statement (first form) or a group of statements (second form) under control of a named variable. The variable takes on the value of the first expression, then is incremented by one on each loop, not to exceed the value of the second expression. The third and fourth forms require three expressions separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action (normally an increment).

fun f([a, ...]) [v, ...]
...
nuf
Fun defines the function name, arguments, and local variables for a user-written function. Up to ten arguments and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions may not be nested.

freturn
A way to signal the failure of a user-written function. See the interrogation operator (**?**) below. If interrogation is not present, *freturn* merely returns zero. When interrogation *is* active, *freturn* transfers to that expression (possibly by-passing intermediate function returns).

goto name
Control is passed to the internally stored statement with the matching label.

ibase *N*
Ibase sets the input base (radix) to *N*. The only supported values for *N* are **8**, **10** (the default), and **16**. Hexadecimal values 10−15 are entered as **a**−**f**. A leading digit is required (i.e., **f0a** must be entered as **0f0a**). *Ibase* (and *obase*, below) are executed immediately.

if expression statement
if expression
...
[**else**
...]

fi

The statement (first form) or group of statements (second form) is executed if the expression evaluates to non-zero. The strings **0** and "" (null) evaluate as zero. In the second form, an optional *else* allows for a group of statements to be executed when the first group is not. The only statement permitted on the same line with an *else* is an *if*; only other *fi*'s can be on the same line with a *fi*. The elision of *else* and *if* into an *elif* is supported. Only a single *fi* is required to close an *if* ... *elif* ... [*else* ...] sequence.

include expression

The expression must evaluate to a file name. The file must contain *bs* source statements. Such statements become part of the program being compiled. *Include* statements may not be nested.

obase *N*

Obase sets the output base to *N* (see *ibase* above).

onintr label
onintr

The *onintr* command provides program control of interrupts. In the first form, control will pass to the label given, just as if a *goto* had been executed at the time *onintr* was executed. The effect of the statement is cleared after each interrupt. In the second form, an interrupt will cause *bs* to terminate.

return [expression]

The expression is evaluated and the result is passed back as the value of a function call. If no expression is given, zero is returned.

run

The random number generator is reset. Control is passed to the first internal statement. If the *run* statement is contained in a file, it should be the last statement.

stop

Execution of internal statements is stopped. *Bs* reverts to immediate mode.

trace [expression]

The *trace* statement controls function tracing. If the expression is null (or evaluates to zero), tracing is turned off. Otherwise, a record of user-function calls/returns will be printed. Each *return* decrements the *trace* expression value.

while expression statement
while expression

...

next

While is similar to *for* except that only the conditional expression for loop-continuation is given.

! shell command

An immediate escape to the Shell.

...

This statement is ignored. It is used to interject commentary in a program.

Expression Syntax:

name
A name is used to specify a variable. Names are composed of a letter (upper or lower case) optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared in *fun* statements, all names are global to the program. Names can take on numeric (double float) values, string values, or can be associated with input/output (see the built-in function *open*( ) below).

name ([expression [, expression] ...])
Functions can be called by a name followed by the arguments in parentheses separated by commas. Except for built-in functions (listed below), the name must be defined with a *fun* statement. Arguments to functions are passed by value.

name [expression [, expression] ...]
This syntax is used to reference either arrays or tables (see built-in *table* functions below). For arrays, each expression is truncated to an integer and used as a specifier for the name. The resulting array reference is syntactically identical to a name; **a[1,2]** is the same as **a[1][2]**. The truncated expressions are restricted to values between 0 and 32767.

number
A number is used to represent a constant value. A number is written in Fortran style, and contains digits, an optional decimal point, and possibly a scale factor consisting of an **e** followed by a possibly signed exponent.

string
Character strings are delimited by " characters. The \ escape character allows the double quote (**\"**), new-line (**\n**), carriage return (**\r**), backspace (**\b**), and tab (**\t**) characters to appear in a string. Otherwise, \ stands for itself.

(expression)
Parentheses are used to alter the normal order of evaluation.

(expression, expression [, expression ...]) [expression]
The bracketed expression is used as a subscript to select a comma-separated expression from the parenthesized list. List elements are numbered from the left, starting at zero. The expression:

(False, True)[a == b]

has the value **True** if the comparison is true.

? expression
The interrogation operator tests for the success of the expression rather than its value. At the moment, it is useful for testing end-of-file (see examples in the *Programming Tips* section below), the result of the *eval* built-in function, and for checking the return from user-written functions (see *freturn*). An interrogation "trap" (end-of-file, etc.) causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

– expression
The result is the negation of the expression.

++ name
Increments the value of the variable (or array reference). The result is the new value.

-- name
Decrements the value of the variable. The result is the new value.

! expression
The logical negation of the expression. Watch out for the shell escape command.

expression *operator* expression
Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. Except for the assignment, concatenation, and relational operators, both operands are converted to numeric form before the function is applied.

Binary Operators (in increasing precedence):

=
= is the assignment operator. The left operand must be a name or an array element. The result is the right operand. Assignment binds right to left, all other operators bind left to right.

_
_ (underscore) is the concatenation operator.

& |
& (logical and) has result zero if either of its arguments are zero. It has result one if both of its arguments are non-zero; | (logical or) has result zero if both of its arguments are zero. It has result one if either of its arguments is non-zero. Both operators treat a null string as a zero.

< <= > >= == !=
The relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, != not equal to) return one if their arguments are in the specified relation. They return zero otherwise. Relational operators at the same level extend as follows: *a*>*b*>*c* is the same as *a*>*b* & *b*>*c*. A string comparison is made if both operands are strings.

+ -
Add and subtract.

* / %
Multiply, divide, and remainder.

^
Exponentiation.

Built-in Functions:

Dealing with arguments

arg(i)
is the value of the *i*-th actual parameter on the current level of function call. At level zero, *arg* returns the *i*-th command-line argument (*arg*(0) returns **bs**).

narg()
returns the number of arguments passed. At level zero, the command argument count is returned.

Mathematical

abs(x)
is the absolute value of x.

atan(x)
is the arctangent of x. Its value is between $-\pi/2$ and $\pi/2$.

ceil(x)
returns the smallest integer not less than x.

cos(x)
is the cosine of x (radians).

exp(x)
is the exponential function of x.

floor(x)
returns the largest integer not greater than x.

log(x)
is the natural logarithm of x.

rand()
is a uniformly distributed random number between zero and one.

sin(x)
is the sine of x (radians).

sqrt(x)
is the square root of x.

String operations

size(s)
the size (length in bytes) of *s* is returned.

format(f, a)
returns the formatted value of *a*. *F* is assumed to be a format specification in the style of *printf*(3S). Only the %...**f**, %...**e**, and %...**s** types are safe.

index(x, y)
returns the number of the first position in *x* that any of the characters from *y* matches. No match yields zero.

trans(s, f, t)
Translates characters of the source *s* from matching characters in *f* to a character in the same position in *t*. Source characters that do not appear in *f* are copied to the result. If the string *f* is longer than *t*, source characters that match in the excess portion of *f* do not appear in the result.

substr(s, start, width)
returns the sub-string of *s* defined by the *start*ing position and *width*.

match(string, pattern)
mstring(n)
The *pattern* is similar to the regular expression syntax of the *ed*(1) command. The characters ., [,], ^ (inside brackets), * and $ are special. The *mstring* function returns the *n*-th ($1 <= n <= 10$) substring of the subject that occurred between pairs of the pattern symbols \(and \) for the most recent call to *match*. To succeed, patterns must match the beginning of the string (as if all patterns began with ^). The function

returns the number of characters matched. For example:

```
match("a123ab123", ".*\([a-z]\)") == 6
mstring(1) == "b"
```

File handling

open(name, file, function)
close(name)
The *name* argument must be a *bs* variable name (passed as a string). For the *open*, the *file* argument may be **1)** a 0 (zero), 1, or 2 representing standard input, output, or error output, respectively, **2)** a string representing a file name, or **3)** a string beginning with an **!** representing a command to be executed (via *sh* *-c*). The *function* argument must be either **r** (read), **w** (write), **W** (write without new-line), or **a** (append). After a *close*, the *name* reverts to being an ordinary variable. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

Examples are given in the following section.

access(s, m)
executes *access*(2).

ftype(s)
returns a single character file type indication: **f** for regular file, **p** for FIFO (i.e., named pipe), **d** for directory, **b** for block special, or **c** for character special.

Tables

table(name, size)
A table in *bs* is an associatively accessed, single-dimension array. "Subscripts" (called keys) are strings (numbers are converted). The *name* argument must be a *bs* variable name (passed as a string). The *size* argument sets the minimum number of elements to be allocated. *Bs* prints an error message and stops on table overflow.

item(name, i)

key()
The *item* function accesses table elements sequentially (in normal use, there is no orderly progression of key values). Where the *item* function accesses values, the *key* function accesses the "subscript" of the previous *item* call. The *name* argument should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table, for example:

```
table("t", 100)
...
# If word contains "party", the following expression adds one
# to the count of that word:
++t[word]
...
# To print out the the key/value pairs:
for i = 0, ?(s = item(t, i)), ++i  if key()  put = key()_":"_s
```

iskey(name, word)
The *iskey* function tests whether the key **word** exists in the table **name** and returns one for true, zero for false.

Odds and ends

eval(s)
The string argument is evaluated as a *bs* expression. The function is handy for converting numeric strings to numeric internal form. *Eval* can also be used as a crude form of indirection, as in:

```
name = "xyz"
eval("++"_ name)
```

which increments the variable *xyz*. In addition, *eval* preceded by the interrogation operator permits the user to control *bs* error conditions. For example:

```
?eval("open(\"X\", \"XXX\", \"r\")")
```

returns the value zero if there is no file named "XXX" (instead of halting the user's program). The following executes a *goto* to the label *L* (if it exists):

```
label="L"
if !(?eval("goto "_ label)) puterr = "no label"
```

plot(request, args)
The *plot* function produces output on devices recognized by *tplot*(1G). The *requests* are as follows:

Call	*Function*
plot(0, term)	causes further *plot* output to be piped into *tplot*(1G) with an argument of −T*term*.
plot(4)	"erases" the plotter.
plot(2, string)	labels the current point with *string*.
plot(3, x1, y1, x2, y2)	draws the line between (*x1*,*y1*) and (*x2*,*y2*).
plot(4, x, y, r)	draws a circle with center (*x*,*y*) and radius *r*.
plot(5, x1, y1, x2, y2, x3, y3)	draws an arc (counterclockwise) with center (*x1*,*y1*) and endpoints (*x2*,*y2*) and (*x3*,*y3*).
plot(6)	is not implemented.
plot(7, x, y)	makes the current point (*x*,*y*).
plot(8, x, y)	draws a line from the current point to (*x*,*y*).
plot(9, x, y)	draws a point at (*x*,*y*).
plot(10, string)	sets the line mode to *string*.
plot(11, x1, y1, x2, y2)	makes (*x1*,*y1*) the lower left corner of the plotting area and (*x2*,*y2*) the upper right corner of the plotting area.

plot(12,x1,y1,x2,y2) causes subsequent x (y) coordinates to be multiplied by *x1* (*y1*) and then added to *x2* (*y2*) before they are plotted. The initial scaling is **plot(12, 1.0, 1.0, 0.0, 0.0)**.

Some requests do not apply to all plotters. All requests except zero and twelve are implemented by piping characters to *tplot*(1G). See *plot*(4) for more details.

last()
in immediate mode, *last* returns the most recently computed value.

PROGRAMMING TIPS

Using *bs* as a calculator:

```
$ bs
#   Distance (inches) light travels in a nanosecond.
186000 * 5280 * 12 / 1e9
11.78496
...

#   Compound interest (6% for 5 years on $1,000).
int = .06 / 4
bal = 1000
for i = 1 5*4  bal = bal + bal*int
bal - 1000
346.855007
...
exit
```

The outline of a typical *bs* program:

```
# initialize things:
var1 = 1
open("read", "infile", "r")
...
# compute:
while ?(str = read)
        ...
next
# clean up:
close("read")
...
# last statement executed (exit or stop):
exit
# last input line:
run
```

Input/Output examples:

```
#   Copy "oldfile" to "newfile".
open("read", "oldfile", "r")
open("write", "newfile", "w")
...
while ?(write = read)
...
# close "read" and "write":
```

```
close("read")
close("write")

#    Pipe between commands.
open("ls", "!ls *", "r")
open("pr", "!pr -2 -h 'List'", "w")
while ?(pr = ls)  ...
...
# be sure to close (wait for) these:
close("ls")
close("pr")
```

EXAMPLE

bs program 1 2 3

compiles and/or executes the file named "program" as well as statements typed from standard input. The arguments "1", "2," and "3" are passed as arguments to the compiled/executed program.

SEE ALSO

ed(1), sh(1), tplot(1G), access(2), printf(3S), stdio(3S), plot(4).
See Section 3 of this volume for further description of the mathematical functions (*pow* on *exp*(3M) is used for exponentiation); *bs* uses the Standard Input/Output package.

NAME

cal – print calendar

SYNOPSIS

cal [month] year

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. *Year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

EXAMPLE

cal 9 1752

produces a calendar for September 1752.

BUGS

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

Beware that "cal 78" refers to the early Christian era, not the 20th century.

NAME

calendar – reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

Calendar consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as "Dec. 7," "december 7," "12/7," etc., are recognized, but not "7 December' or "7/12". On weekends "tomorrow" extends through Monday.

When an argument is present, *calendar* does its job for every user who has a file **calendar** in their login directory and sends them any positive results by *mail*(1). Normally this is done daily by facilities in the UNIX operating system under control of *cron*(1M).

EXAMPLE

If the user has the following line, among other lines containing date information, in the file "calendar" in the login directory:

Monday, September 6 Labor Day Holiday

typing in

calendar

either on the Friday before or on the specified Monday will cause this line to be printed on the screen.

FILES

calendar
/usr/lib/calprog to figure out today's and tomorrow's dates
/etc/passwd
/tmp/cal*
/usr/lib/crontab

SEE ALSO

mail(1).

BUGS

Your calendar must be public information for you to get reminder service. *Calendar's* extended idea of "tomorrow" does not account for holidays.

NAME

cat − concatenate and print files

SYNOPSIS

cat [**−u**] [**−s**] file ...

DESCRIPTION

Cat reads each *file* in sequence and writes it on the standard output.

If no input file is given, or if the argument − is encountered, *cat* reads from the standard input file. Output is buffered unless the **−u** option is specified. The **−s** option makes *cat* silent about non-existent files. No input file may be the same as the output file unless it is a special file.

EXAMPLE

```
cat file
```

prints the file, and:

```
cat file1 file2 > file3
```

concatenates the first two files and places the result in the third.

WARNING

Command formats such as

```
cat file1 file2 >file1
```

will cause the original data in *file1* to be lost, therefore, take care when using shell special characters.

SEE ALSO

cp(1), pr(1).

NAME

cb − C program beautifier

SYNOPSIS

cb [**−s**] [**−j**] [**−l** leng] [file ...]

DESCRIPTION

Cb reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code. Under default options, *cb* preserves all user new-lines. Under the **−s** flag *cb* canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*. The **−j** flag causes split lines to be put back together. The **−l** flag causes *cb* to split lines that are longer than *leng*.

EXAMPLE

If there is a C program called *test.c* which looks like this:

```
#define COMING 1
#define GOING 0

main ()
{
/* This is a test of the C Beautifier */
if (COMING)
printf ("Hello, world\n");
else
printf ("Goodbye, world\n");
}
```

Then using the *cb* command as shown below produces the output shown:

```
cb test.c
#define COMING 1
#define GOING 0

main ()
{
        /* This is a test of the C Beautifier */
        if (COMING)
                printf ("Hello, world\n");
        else
                printf ("Goodbye, world\n");
}
```

SEE ALSO

cc(1).

The C Programming Language by B. W. Kernighan and D. M. Ritchie.

BUGS

Punctuation that is hidden in preprocessor statements will cause indentation errors.

NAME

cc − C compiler

SYNOPSIS

cc [option] ... file ...

DESCRIPTION

Cc is the UNIX C compiler.

Cc accepts several types of arguments:

Arguments whose names end with '.c' are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with '.o' substituted for '.c'. The '.o' file is normally deleted if a single C program is compiled and loaded.

In the same way, arguments whose names end with '.s' are taken to be assembly source programs and are assembled, producing a '.o' file.

The following options are interpreted by *cc*. See *ld*(1) for link editor options.

−c Suppress the link edit phase of the compilation, and force an object file to be produced even if only one program is compiled.

−n Passed on to *ld* to make the text of the resulting program shared.

−p Arrange for the compiler to produce code which counts the number of times each routine is called; also, if link editing takes place, replace the standard startup routine by one which automatically calls *monitor*(3C) at the start and arranges to write out a *mon.out* file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1).

−fsky Use the sky floating point library.

−O(BKPS)
Invoke an object-code improver (optimizer). If **B** is specified, "jump to subroutine" instructions are changed to "branch to subroutine" instructions (where possible). If **K** is specified, certain UNIX kernel optimizer functions are not performed. If **P** is specified, stack probe instructions are removed. (Note: **P** should only be used for the operating system source.) If **S** is specified, stack frame optimization is performed and the debugger, *adb*(1), might indicate too few subroutine parameters on stack trace back.

−R (addr)
Pass on to *ld*, making the resulting object module *origin*ed at *addr(hex)*.

−S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed '.s'.

−E Run only *cpp*(1) on the named C programs, and send the result to standard output.

−P Run only the macro preprocessor on the named C programs, and send the result to the corresponding files suffixed '.i'

-C Prevent the macro preprocessor from eliding (leaving out) comments.

-o *output* Name the final executable output file *output*. If this option is used the file "a.out" will be left undisturbed.

-D*name*=*def*

-D*name* Define the *name* to the preprocessor, as if by **#define**. If no definition is given, the name is defined as "1".

-U*name* Remove any initial definition of *name*.

-I*dir* **#include** files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **-I** options, then in the directory **/usr/include**.

-v print the name of each subprocess as it is executing.

Other arguments are taken to be either link editor option arguments, or C-compatible object programs, typically produced by an earlier *cc* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked via *ld*(1) (in the order given) to produce an executable program with name *a.out*.

EXAMPLE

```
cc -o output prog1.c prog2.c prog3.c
```

would compile code in the three named C programs and put the compiled code into the file "output".

FILES

file.c	input file
file.o	object file
a.out	linked output
/tmp/ctm?	temporary
/lib/cpp	preprocessor
/lib/c	combined compiler pass1 and pass2
/lib/c0	compiler pass1
/lib/c1	compiler pass2
/lib/c2	optional optimizer invoked with "-O"
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	runtime startoff for profiling
/lib/libc.a	standard library, see section 3
/usr/include	standard directory for '#include' files
/lib/libm.a	math library
/lib/libsky.a	sky floating point routines
/lib/crt0sky.o	runtime startoff using sky
/lib/mcrt0sky.o	runtime startoff for profiling using sky

SEE ALSO

adb(1), ld(1), lint(1), prof(1), monitor(3C)
The C Programming Language, Prentice-Hall, 1978, by B. W. Kernighan and D. M. Ritchie; *Programming in C—a tutorial*, by B. W. Kernighan; *C Reference Manual*, by D. M. Ritchie

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or the link editor. Confusing syntax may cause the C compiler to indicate an error on the line following the actual error.

NAME

cd – change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

If *directory* is not specified, the value of shell parameter **$HOME** is used as the new working directory. If *directory* specifies a complete path starting with /, ., .., *directory* becomes the new working directory. If neither case applies, *cd* tries to find the designated directory relative to one of the paths specified by the **$CDPATH** shell variable. **$CDPATH** has the same syntax as, and similar semantics to, the **$PATH** shell variable. *Cd* must have execute (search) permission in *directory*.

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command; therefore, it is recognized and internal to the shell.

EXAMPLE

```
cd /unisoft/usr/games
```

would relocate you to the directory **/unisoft/usr/games** if this directory is executable (searchable) by you.

SEE ALSO

pwd(1), sh(1), chdir(2).

NAME

cdc − change the delta commentary of an SCCS delta

SYNOPSIS

cdc −r*SID* [−**m**[*mrlist*]] [−**y**[*comment*]] *files*

DESCRIPTION

Cdc changes the *delta commentary*, for the *SID* specified by the −**r** keyletter, of each named SCCS file.

Delta commentary is defined to be the Modification Request (**MR**) and comment information normally specified via the *delta*(1) command (−**m** and −**y** keyletters).

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

Arguments to *cdc*, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

−r*SID* Used to specify the *S*CCS *ID*entification (*SID*) string of a delta for which the delta commentary is to be changed.

−**m**[*mrlist*] If the SCCS file has the **v** flag set (see *admin*(1)) then a list of **MR** numbers to be added and/or deleted in the delta commentary of the *SID* specified by the −**r** keyletter *may* be supplied. A null **MR** list has no effect.

MR entries are added to the list of **MR**s in the same manner as that of *delta*(1). In order to delete an **MR**, precede the **MR** number with the character **!** (see *EXAMPLE*). If the **MR** to be deleted is currently in the list of **MR**s, it is removed and changed into a "comment" line. A list of all deleted **MR**s is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If −**m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see −**y** keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MR** list.

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell

procedure) which validates the correctness of the **MR** numbers. If a non-zero exit status is returned from the **MR** number validation program, *cdc* terminates and the delta commentary remains unchanged.

−y[*comment*] Arbitrary text used to replace the *comment*(s) already existing for the delta specified by the **−r** keyletter. The previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If **−y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the *comment* text.

The exact permissions necessary to modify the SCCS file are documented in the *Source Code Control System User's Guide*. Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLE

```
cdc −r1.6 −m"bl78-12345 !bl77-54321 bl79-00001" −ytrouble
s.file
```

adds bl78-12345 and bl79-00001 to the **MR** list, removes bl77-54321 from the **MR** list, and adds the comment **trouble** to delta 1.6 of s.file.

```
cdc −r1.6 s.file
MRs? !bl77-54321 bl78-12345 bl79-00001
comments? trouble
```

does the same thing.

WARNINGS

If SCCS file names are supplied to the *cdc* command via the standard input (− on the command line), then the **−m** and **−y** keyletters must also be used.

FILES

x-file (see *delta*(1))
z-file (see *delta*(1))

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4).
"Source Code Control System User's Guide"

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

cflow − generate C flow graph

SYNOPSIS

cflow [−**r**] [−**ix**] [−**i_**] [−**d**num] files

DESCRIPTION

Cflow analyzes a collection of C, YACC, LEX, assembler, and object files and attempts to build a graph charting the external references. Files suffixed in **.y**, **.l**, **.c**, and **.i** are YACC'd, LEX'd, and C-preprocessed (bypassed for **.i** files) as appropriate and then run through the first pass of *lint*(1). (The −**I**, −**D**, and −**U** options of the C-preprocessor are also understood.) Files suffixed with **.s** are assembled and information is extracted (as in **.o** files) from the symbol table. The output of all this non-trivial processing is collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference (i.e., line) number, followed by a suitable number of tabs indicating the level. Then the name of the global (normally only a function not defined as an external or beginning with an underscore; see below for the −**i** inclusion option) a colon and its definition. For information extracted from C source, the definition consists of an abstract type declaration (e.g., **char** *), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (e.g., *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only < > is printed.

When the nesting level becomes too deep, the −**e** option of *pr*(1) can be used to compress the tab expansion to something less than every eight spaces.

The following options are interpreted by *cflow*:

−**r** Reverse the "caller:callee" relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.

−**ix** Include external and static data symbols. The default is to include only functions in the flow graph.

−**i_** Include names that begin with an underscore. The default is to exclude these functions (and data if *-ix* is used).

−**d**num The *num* decimal integer indicates the depth at which the flow graph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be met with contempt.

EXAMPLE

Given the following in "file.c":

```
int     i;
main()
{
        f();
        g();
        f();
}
f()
{
        i = h();
}
```

the command:

```
cflow file.c
```

produces the the output:

```
1       main: int(), <file.c 4>
2               f: int(), <file.c 11>
3                       h: <>
4               g: <>
```

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C-preprocessor).

SEE ALSO

as(1), cc(1), lex(1), lint(1), nm(1), pr(1), yacc(1).

BUGS

Files produced by *lex*(1) and *yacc*(1) cause the reordering of line number declarations which can confuse *cflow*. To get proper results, feed *cflow* the *yacc* or *lex* input.

NAME

chmod − change mode

SYNOPSIS

chmod mode files

DESCRIPTION

The permissions of the named *files* are changed according to *mode*, which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, see *chmod*(2)
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*]

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, the default if *who* is omitted.

Op can be + to add *permission* to the file's mode, − to take away *permission*, or = to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group ID) and **t** (save text, or sticky); **u**, **g**, or **o** indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with = to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter **s** is only useful with **u** or **g** and **t** only works with **u**.

Only the owner of a file (or the super-user) may change its mode.

EXAMPLE

chmod 755 filename

changes the mode of "filename" to: read, write, execute (400+200+100) by owner; read, execute (40+10) for group; read, execute (4+1) for others. An *ls -l* of filename shows [-rwxr-xr-x filename] that the requested mode is in effect.

chmod = filename

will take away all permissions from *filename*, including yours.

chmod o-w file

denies write permission to others.

chmod +x file

makes a file executable.

SEE ALSO

ls(1), chmod(2).

NAME

chown, chgrp – change owner or group

SYNOPSIS

chown owner file ...

chgrp group file ...

DESCRIPTION

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

Chgrp changes the group ID of the *files* to *group*. The group may be either a decimal group ID or a group name found in the group file.

EXAMPLE

```
chown unisoft filea fileb filec
```

would make "unisoft" the owner of the three files.

FILES

/etc/passwd
/etc/group

SEE ALSO

chown(2), group(4), passwd(4).

NAME

clear – clear terminal screen

SYNOPSIS

clear

DESCRIPTION

Clear clears your screen if this is possible. It looks in the environment for the terminal type (**TERM**) and capabilities string (**TERMCAP**). If **TERMCAP** is not found in the environment, it looks in **/etc/termcap** to figure out how to clear the screen.

EXAMPLE

```
clear
```

clears the screen.

FILES

/etc/termcap terminal capability data base

SEE ALSO

environ(4), termcap(5)

NAME

cmp − compare two files

SYNOPSIS

cmp [**−l**] [**−s**] file1 file2

DESCRIPTION

The two files are compared. (If *file1* is −, the standard input is used.) Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

Options:

−l Print the byte number (decimal) and the differing bytes (octal) for each difference.

−s Print nothing for differing files; return codes only.

EXAMPLE

```
cmp alpha beta
```

will report if the files are different and at what point they differ, such as:

```
alpha beta differ: char 33, line 2
```

SEE ALSO

comm(1), diff(1).

DIAGNOSTICS

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

NAME

col − filter reverse line-feeds

SYNOPSIS

col [−bfpx]

DESCRIPTION

Col reads from the standard input and writes onto the standard output. It performs the line overlays implied by reverse line feeds (ASCII code ESC-7), and by forward and reverse half-line-feeds (**ESC-9** and **ESC-8**). *Col* is particularly useful for filtering multicolumn output made with the **.rt** command of *nroff* and output resulting from use of the *tbl*(1) preprocessor.

If the **−b** option is given, *col* assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, only the last one read will be output.

Although *col* accepts half-line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. This treatment can be suppressed by the **−f** (fine) option; in this case, the output from *col* may contain forward half-line-feeds (**ESC-9**), but will still never contain either kind of reverse line motion.

Unless the **−x** option is given, *col* will convert white space to tabs on output wherever possible to shorten printing time.

The ASCII control characters **SO** (\017) and **SI** (\016) are assumed by *col* to start and end text in an alternate character set. The character set to which each input character belongs is remembered, and on output **SI** and **SO** characters are generated as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab, return, new-line, **SI**, **SO**, **VT** (\013), and **ESC** followed by **7**, **8**, or **9**. The **VT** character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

Normally, *col* will ignore any unknown to it escape sequences found in its input; the **−p** option may be used to cause *col* to output these sequences as regular characters, subject to overprinting from reverse line motions. The use of this option is highly discouraged unless the user is fully aware of the textual position of the escape sequences.

EXAMPLE

```
nroff -ms filea | col
```

pipes multicolumn *nroff* output through the *col* filter to enable proper creation of columns.

SEE ALSO

nroff(1), tbl(1).

NOTES

The input format accepted by *col* matches the output produced by *nroff* with either the **−T37** or **−Tlp** options. Use **−T37** (and the **−f** option of *col*) if the ultimate disposition of the output of *col* will be a device that can interpret half-line motions, and **−Tlp** otherwise.

BUGS

Cannot back up more than 128 lines.
Allows at most 800 characters, including backspaces, on a line.
Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

NAME

comb − combine SCCS deltas

SYNOPSIS

comb [−**o**] [−**s**] [−**p**sid] [−**c**list] files

DESCRIPTION

Comb generates a shell procedure (see *sh*(1)) which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the 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 generated shell procedure is written on the standard output.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

−**p***SID* The *S*CCS *ID*entification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

−**c***list* A *list* (see *get*(1) for the syntax of a *list*) of deltas to be preserved. All other deltas are discarded.

−**o** For each **get** −**e** generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the −**o** keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

−**s** This argument causes *comb* to generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

100 * (original − combined) / original

It is recommended that before any SCCS files are actually combined, one should use this option to determine exactly how much space is saved by the combining process.

If no keyletter arguments are specified, *comb* will preserve only leaf deltas and the minimal number of ancestors needed to preserve the tree.

EXAMPLE

comb s.file1 > tmp1

produces a shell script saved in "tmp1" which will remove from the SCCS-format file, "s.file1", all deltas previous to the last set of changes, i.e., removes the capability to return to earlier versions.

FILES

s.COMB The name of the reconstructed SCCS file.
comb????? Temporary.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4).
"Source Code Control System User's Guide"

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

NAME

comm − select or reject lines common to two sorted files

SYNOPSIS

comm [− [**123**]] file1 file2

DESCRIPTION

Comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort*(1)), and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name − means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus **comm −12** prints only the lines common to the two files; **comm −23** prints only lines in the first file but not in the second; **comm −123** is a no-op.

EXAMPLE

```
comm -12 filea fileb
```

prints only the lines common to filea and fileb.

```
comm -23 filea fileb
```

prints only lines in the first file but not in the second.

```
comm -123 filea fileb
```

is not an option, as it suppresses all output.

```
comm -3 filea fileb
```

prints only the lines that differ in the two files.

SEE ALSO

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

NAME

cp, ln, mv — copy, link or move files

SYNOPSIS

cp file1 [file2 ...] target
ln file1 [file2 ...] target
mv file1 [file2 ...] target

DESCRIPTION

File1 is copied (linked, moved) to *target*. Under no circumstance can *file1* and *target* be the same (take care when using *sh*(1) metacharacters). If *target* is a directory, then one or more files are copied (linked, moved) to that directory.

If *mv* determines that the mode of *target* forbids writing, it will print the mode (see *chmod*(2)) and read the standard input for one line (if the standard input is a terminal); if the line begins with **y**, the move takes place; if not, *mv* exits.

Only *mv* will allow *file1* to be a directory, in which case the directory rename will occur only if the two directories have the same parent.

EXAMPLE

```
cp alpha beta gamma /unisoft/roxanne
```

places copies of the three files in the directory **/unisoft/roxanne**.

SEE ALSO

cpio(1), rm(1), chmod(2).

BUGS

If *file1* and *target* 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.

Ln will not link across file systems.

NAME

cpio − copy file archives in and out

SYNOPSIS

cpio −o [**acBv**]

cpio −i [**BcdmrtuvfsSb6**] [patterns]

cpio −p [**adlmruv**] directory

DESCRIPTION

Cpio −o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information.

Cpio −i (copy in) extracts files from the standard input which is assumed to be the product of a previous **cpio −o**. Only files with names that match *patterns* are selected. *Patterns* are given in the name-generating notation of *sh*(1). In *patterns*, meta-characters ?, *, and [...] match the slash / character. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (i.e., select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below.

Cpio −p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

The meanings of the available options are:

a Reset access times of input files after they have been copied.
B Input/output is to be blocked 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from **/dev/rmt?**).
d *Directories* are to be created as needed.
c Write *header* information in ASCII character form for portability.
r Interactively *rename* files. If the user types a null line, the file is skipped.
t Print a *table of contents* of the input. No files are created.
u Copy *unconditionally* (normally, an older file will not replace a newer file with the same name).
v *Verbose*: causes a list of file names to be printed. When used with the **t** option, the table of contents looks like the output of an **ls −l** command (see *ls*(1)).
l Whenever possible, link files rather than copying them. Usable only with the **−p** option.
m Retain previous file modification time. This option is ineffective on directories that are being copied.
f Copy in all files except those in *patterns*.
s Swap bytes. Use only with the **−i** option.
S Swap halfwords. Use only with the **−i** option.
b Swap both bytes and halfwords. Use only with the **−i** option.
6 Process an old (i.e., UNIX System *Sixth* Edition format) file. Only useful with **−i** (copy in).

EXAMPLE

```
ls | cpio −o >/dev/mt0
```

copies the contents of a directory into an archive;

```
cd olddir
find . -depth -print | cpio -pdl newdir
```

duplicates a directory hierarchy.

The trivial case "find . -depth -print | cpio -oB >/dev/rmt0" can be handled more efficiently by:

```
find . -cpio /dev/rmt0
```

SEE ALSO

ar(1), find(1), cpio(4).

BUGS

Path names are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost. Only the super-user can copy special files. The **-B** option does not work with certain magnetic tape drives.

NAME

cpp − the C language preprocessor

SYNOPSIS

/lib/cpp [option ...] [ifile [ofile]]

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc*(1) command. Thus the output of *cpp* is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, *cpp* and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of *cpp* other than in this framework is not suggested. The preferred way to invoke *cpp* is through the *cc*(1) command since the functionality of *cpp* may someday be moved elsewhere. See *m4*(1) for a general macro processor.

Cpp optionally accepts two file names as arguments. *Ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following *options* to *cpp* are recognized:

−P Preprocess the input without producing the line control information used by the next pass of the C compiler and leave the result in *ifile.o*.

−E Preprocess the input and put the resulting output on the standard output.

−C By default, *cpp* strips C-style comments. If the **−C** option is specified, all comments (except those found on *cpp* directive lines) are passed along.

−U*name*
Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

operating system:	ibm, gcos, os, tss, unix
hardware:	interdata, m68000, pdp11, u370, u3b, vax
UNIX System variant:	RES, RT

−D*name*
−D*name*=*def*
Define *name* as if by a **#define** directive. If no =*def* is given, *name* is defined as 1.

−I*dir* Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in "" will be searched for first in the directory of the *ifile* argument, then in directories named in **−I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the *ifile* argument is not searched.

Two special names are understood by *cpp*. The name **__LINE__** is defined as the current line number (as a decimal integer) as known by *cpp*, and **__FILE__** is defined as the current file name (as a C string) as known by *cpp*. They can be used anywhere (including in macros) just as any other defined name.

All *cpp* directives start with lines begun by **#**. The directives are:

#define *name token-string*
Replace subsequent instances of *name* with *token-string*.

#define *name*(*arg*, ..., *arg*) *token-string*
Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma separated tokens, and a) by *token-string* where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma separated list.

#undef *name*
Cause the definition of *name* (if any) to be forgotten from now on.

#include "*filename*"
#include <*filename*>
Include at this point the contents of *filename* (which will then be run through *cpp*). When the <*filename*> notation is used, *filename* is only searched for in the standard places. See the **-I** option above for more detail.

#line *integer-constant* "*filename*"
Causes *cpp* to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file where it comes from. If "*filename*" is not given, the current file name is unchanged.

#endif
Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

#ifdef *name*
The lines following will appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#ifndef *name*
The lines following will not appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#if *constant-expression*
Lines following will appear in the output if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the **?:** operator, the unary **-**, **!**, and **~** operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *cpp* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

#else
Reverses the notion of the test directive which matches this directive. So if lines previous to this directive are ignored, the following lines will appear in the output. And vice versa.

The test directives and the possible **#else** directives can be nested.

EXAMPLE

/lib/cpp −P −DXYZ −DMYFILE=myfile −I../include myprog.c myprog.i

would preprocess "myprog.c" input output file "myprog.i", deleting output line numbers (**−P**), defining symbol XYZ to be null, symbol MYFILE to be "myfile" and using include files from **../include**.

FILES

/usr/include standard directory for **#include** files

SEE ALSO

cc(1), m4(1).

DIAGNOSTICS

The error messages produced by *cpp* are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NOTES

When newline characters were found in argument lists for macros to be expanded, previous versions of *cpp* put out the newlines as they were found and expanded. The current version of *cpp* replaces these newlines with blanks to alleviate problems that the previous versions had when this occurred.

NAME

crypt — encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

Crypt reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. *Crypt* encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

will print the clear.

Files encrypted by *crypt* are compatible with those treated by the editor *ed* in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; "sneak paths" by which keys or clear text can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e. to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lowercase letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. The choice of keys and key security are the most vulnerable aspect of *crypt*.

EXAMPLE

```
crypt asa < sleeper.c > zzz
```

will use the string "asa" as key to the encryption algorithm to encrypt the contents of "sleeper.c", and place the encrypted output in file "zzz". File "zzz" at this point will be unreadable. NOTE that the original file, "sleeper.c", remains in readable form. To obtain readable print-out of the file "zzz", it could be decoded as follows:

```
crypt < zzz
```

After the response:

```
Enter key:
```

the user types in "asa".

FILES

/dev/tty for typed key

SEE ALSO

ed(1), makekey(1).

BUGS

If output is piped to *nroff* and the encryption key is *not* given on the command line, *crypt* can leave terminal modes in a strange state (see *stty*(1)).
If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the contents of the first of the original files will be decrypted correctly.

NOTE

This utility is not provided with international distribution.

NAME

csh − a shell (command interpreter) with C-like syntax

SYNOPSIS

csh [**−cefinstvVxX**] [arg ...]

DESCRIPTION

Csh is a command language interpreter incorporating a history mechanism (see **History Substitutions**) and a C-like syntax.

An instance of *csh* begins by executing commands from the file ".cshrc" in the *home* directory of the invoker. If this is a login shell, then it also executes commands from the file ".login" there. It is typical for users on CRTs to put the command **stty crt** in their ".login" file, and to also invoke *tset*(1) there.

In the normal case, the shell will then begin reading commands from the terminal, prompting with "%". Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and then parsed. Finally each command in the current line is executed.

When a login shell terminates, it executes commands from the file ".logout" in the user's home directory.

Lexical Structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters &, |, ;, <, >, (,), form separate words. If doubled in &&, ||, << or >>, these pairs form single words. These parser metacharacters may be made part of other words, or their special meaning may be prevented, by preceding them with a backslash (\). A newline preceded by a \ is equivalent to a blank. It is usually necessary to use the backslash to *escape* the parser metacharacters when you want to use them literally rather than as metacharacters.

Strings enclosed in matched pairs of quotation marks, either single or double quotation marks, ', ` or ", form parts of a word. Metacharacters in these strings, including blanks and tabs, do not form separate words. Such quotations have semantics to be described subsequently.

Within pairs of single or double quotation marks, a newline (carriage return) preceded by a \ gives a true newline character. This is used to set up a file of strings separated by newlines, as for *fgrep*(1).

When the shell's input is not a terminal, the character # introduces a comment which continues to the end of the input line. It is prevented from having this special meaning when preceded by \ or if bracketed by a pair of single or double quotation marks.

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed.

A simple command or a sequence of simple commands separated by | characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next.

Sequences of pipelines may be separated by ;, and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an &, which means "run it in background".

Parentheses (and) around a pipeline or sequence of pipelines cause the whole series to be treated as a simple command, which may in turn be a component of a pipeline, etc. It is also possible to separate pipelines with || or && indicating, as in the C language, that the second is to be executed only if the first fails or succeeds, respectively. (See *Expressions*.)

Process ID Numbers

When a process is run in background with &, the shell prints a line which looks like:

1234

indicating that the process which was started asynchronously was number 1234.

Status Reporting

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work.

To check on the status of a process, use the *ps* (process status) command.

Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur.

History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence.

History substitutions begin with the character ! and may begin **anywhere** in the input stream (with the proviso that they **do not** nest.)

This ! may be preceded by a \ to turn off its special meaning; for convenience, a ! is also passed unchanged when it is followed by a blank, tab, newline, = or (.

Therefore, **do not** put a space after the ! and the command reference when you are invoking the shell's history mechanism. (History substitutions also occur when an input line begins with ↑. This special abbreviation will be described later.)

An input line which invokes history substitution is echoed on the terminal before it is executed, as it would look if typed out in full.

The shell's history list, which may be seen by typing the *history* command, contains all commands input from the terminal which consist of one or more words. History substitutions reintroduce sequences of words from these saved commands into the input stream. The *history* variable controls the size of the input stream. The previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

Consider the following output from the *history* command:

```
 9 write michael
10 ex write.c
11 cat oldwrite.c
12 diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing an ! in the prompt string. This is done by SETting Prompt = ! and the prompt character of your choice.

For example, if the current event is number 13, we can call up the command recorded as event 11 in several ways: **!-2** [i.e., 13-2]; by the first letter of one of its command words, such as **!c** referring to the "c" in *cat*; or **!wri** for event 9, or by a string contained in a word in the command as in **!?mic?** also referring to event 9.

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case **!!** refers to the previous command; thus **!!** alone is essentially a *redo*.

Words are selected from a command event and acted upon according to the following formula:

event:position:action

The *event* is the command you wish to retrieve. As mentioned above, it may be summoned up by event number and in several other ways. All that the *event* notation does is to tell the shell which command you have in mind.

Position picks out the words from the command event on which you want the *action* to take place. The *position* notation can do anything from altering the command completely to making some very minor substitution, depending on which words from the command event you specify with the *position* notation.

To select words from a command event, follow the event specification with a : and a designator (by position) for the desired words.

The words of a command event are picked out by their position in the input line. Positions are numbered from 0, the first word (usually command) being position 0, the second word having position 1, and so forth. If you designate a word from the command event by stating its position, means you want to include it in your revised command. All the words that you want to include in a revised command must be designated by position notation in order to be included.

The basic position designators are:

0	first (command) word
n	*n*th argument
↑	first argument, i.e., 1
$	last argument
%	matches the word of an **?s?** search which immediately precedes it; used to strip one word out of a command event for use in another command. Example: **!?four?:%:p** prints **four**.

x−*y* range of words (e.g., 1-3 means "from position 1 to position 3").
−*y* abbreviates "0−*y*
* stands for "↑−$", or indicates position 1 if only one word in event.
*x** abbreviates "*x*−$" where *x* is a position number.
x− like "*x**" but omitting last word "$"

The : separating the event specification from the word designator can be omitted if the argument selector begins with a ↑, $, *, − or %.

Modifiers, each preceded by a :, may be used to act on the designated words in the specified command event. The following modifiers are defined:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing ".xxx" component, leaving the root name.
e	Remove all but the extension ".xxx" part.
s/*old*/*new*/	Substitute *new* for *old*
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g., "g&".
p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like q, but break into words at blanks, tabs and newlines.

Unless preceded by a "g", the modification is applied only to the first modifiable word. With substitutions it is an error for no word to be applicable.

The left hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of /; a \ quotes the delimiter into the *l* and *r* strings. The character & in the right hand side is replaced by the text from the left. A \ quotes & also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in !?*s*?. The trailing delimiter in the substitution may be omitted if (but only if) a newline follows immediately as may the trailing ? in a contextual scan.

A history reference may be given without an event specification, e.g., **!$**. In this case the reference is to the previous command. If a previous history reference occurred on the same line, this form repeats the previous reference. Thus **!?foo?↑ !$** gives the first and last arguments from the command matching **?foo?**.

You can quickly make substitutions to the previous command line by using the ↑ character as the first non-blank character of an input line. This is equivalent to **!:s↑** providing a convenient shorthand for substitutions on the text of the previous line. Thus **↑lb↑lib** fixes the spelling of "lib" in the previous command. Finally, a history substitution may be surrounded with { and } if necessary to insulate it from the characters which follow. Thus, after **ls −ld ~paul** we might do **!{l}a** to do **ls −ld ~paula**, while **!la** would look for a command starting **la**.

Quotations with ' and "

The quotation of strings by ' and " can be used to prevent all or some of the remaining substitutions which would otherwise take place if these characters were interpreted as "metacharacters" or "wild card matching characters". Strings enclosed in single quotes, ' are prevented any further interpretation or expansion. Strings enclosed in " may still be variable and command expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see *Command Substitution* below) does a " quoted string yield parts of more than one word; ' quoted strings never do.

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for *ls* is **ls −l** the command **ls /usr** would map to **ls−l/usr**, the argument list here being undisturbed. Similarly if the alias for *lookup* was **grep !↑ /etc/passwd**, then **lookup bill** would map to **grep bill /etc/passwd**.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can **alias print 'pr \!* | lpr'** to make a command which *pr*s its arguments to the line printer.

Variable substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the *set* and *unset* commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The setting of this variable results from the **−v** command line option.

Other operations treat variables numerically. The @ command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by $ characters. This expansion can be prevented by preceding the $ with a \ except within double quotes (") where it **always** occurs, and within single quotes (') where it **never** occurs. Strings quoted by ` are interpreted later (see *Command substitution* below) so $ substitution does not occur there until later, if at all. A $ is passed unchanged if followed by a blank, tab, or end-of-line.

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

Unless enclosed in double quotes or given the :q modifier, the results of variable substitution may eventually be command and filename substituted. Within double quotes, a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variables value separated by blanks. When the :q modifier is applied to a substitution, the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

Metasequences for variable substitution

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$name
${name}
: Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.

  If *name* is not a shell variable, but is set in the environment, then that value is returned (but : modifiers and the other forms given below are not available in this case).

$name[selector]
${name[selector]}
: May be used to select only some of the words from the value of *name*. The selector is subjected to $ substitution and may consist of a single number or two numbers separated by a −. The first word of a variables value is numbered "1". If the first number of a range is omitted it defaults to "1". If the last member of a range is omitted it defaults to "$#name". The selector * selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$#name
${#name}
: Gives the number of words in the variable. This is useful for later use in a "[selector]".

$0
: Substitutes the name of the file from which command input is being

read. An error occurs if the name is not known.

$number
${number}
Equivalent to "$argv[number]".

$*
Equivalent to "$argv[*]".

The modifiers ":h", ":t", ":r", ":q" and ":x" may be applied to the substitutions above as may ":gh", ":gt" and ":gr". If braces { } appear in the command form, then the modifiers must appear within the braces. The current implementation allows only one : modifier on each $ expansion.

The following substitutions may not be modified with : modifiers.

$?name
${?name}
Substitutes the string "1" if name is set, "0" if it is not.

$?0
Substitutes "1" if the current input filename is known, "0" if it is not.

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

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution

Command substitution is indicated by a command enclosed in `. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within double quotes ("), only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename substitution

If a word contains any of the characters *, ?, [or { or begins with the character ~, then that word is a candidate for filename substitution, also known as "globbing". This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters *, ? and [imply pattern matching, the characters ~ and { being more akin to abbreviations.

In matching filenames, the character . at the beginning of a filename or immediately following a /, as well as the character / must be matched explicitly. The character * matches any string of characters, including the

null string. The character ? matches any single character. The sequence [...] matches any one of the characters enclosed. Within [...], a pair of characters separated by − matches any character lexically between the two.

The character ~ at the beginning of a filename is used to refer to home directories. Standing alone, i.e., ~ it expands to the invokers home directory as reflected in the value of the variable *home*. When followed by a name consisting of letters, digits and − characters, the shell searches for a user with that name and substitutes their home directory; thus **~ken** might expand to **/usr/ken** and **~ken/chmach** to **/usr/ken/chmach**. If the character ~ is followed by a character other than a letter or / or appears not at the beginning of a word, it is left undisturbed.

The metanotation **a{b,c,d}e** is a shorthand for **abeaceade**. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus **~source/s1/{oldls,ls}.c** expands to **/usr/source/s1/oldls.c /usr/source/s1/ls.c** whether or not these files exist without any chance of error if the home directory for **source** is **/usr/source**. Similarly **../{memo,*box}** might expand to **../memo ../box ../mbox**. (Note that "memo" was not sorted with the results of matching "*box".) As a special case {, } and {} are passed undisturbed.

Input/output

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

< name
: Open file *name* (which is first variable, command and filename expanded) as the standard input.

<< word
: Read the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting \, ", ' or ` appears in *word*, variable and command substitution is performed on the intervening lines, allowing \ to quote $, \ and `. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name
>! name
>& name
>&! name
: The file *name* is used as standard output. If the file does not exist then it is created; if the file exists, it is truncated, its previous contents being lost.

  If the variable *noclobber* is set, then the file must not exist or be a character special file (e.g., a terminal or **/dev/null**) or an error results. This helps prevent accidental destruction of files. In this case the ! forms can be used and suppress this check.

  The forms involving &, route the diagnostic output into the specified file as well as the standard output. *Name* is expanded in

the same way as < input filenames are.

>> name
>>& name
>>! name
>>&! name
Uses file *name* as standard output like > but places output at the end of the file. If the variable *noclobber* is set, then it is an error for the file not to exist unless one of the ! forms is given. Otherwise similar to >.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The << mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input.

Diagnostic output may be directed through a pipe with the standard output. Simply use the form |& rather than just |.

Expressions

A number of the builtin commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the @, *exit*, *if*, and *while* commands. The following operators are available:

|| && | ↑ & == != =~ !~ <= >= < > << >> + -
* / % ! ~ ( )

Here the precedence increases to the right, ==, !=, =~ and !~; <=, >=, < and >; << and >>; + and −; *, / and % being, in groups, at the same level. The ==, !=, =~ and !~ operators compare their arguments as strings; all others operate on numbers. The operators =~ and !~ are like != and == except that the right hand side is a *pattern* (containing, e.g., *s, ?s and instances of [...]) against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings which begin with "0" are considered octal numbers. Null or missing arguments are considered "0". The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser (& | < > ( )) they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in { and } and file enquiries of the form **−l name** where **l** is one of:

r read access
w write access
x execute access
e existence
o ownership

z zero size
f plain file
d directory

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible, then all enquiries return false, i.e., "0". Command executions succeed, returning true, i.e., "1", if the command exits with status 0, otherwise they fail, returning false, i.e., "0". If more detailed status information is required, then the command should be executed outside of an expression and the variable *status* examined.

Control Flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The *foreach*, *switch*, and *while* statements, as well as the *if*–*then*–*else* form of the *if* statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward *goto*s will succeed on non-seekable inputs.)

Builtin Commands

Builtin commands are executed within the shell. If a builtin command occurs as any component of a pipeline except the last, then it is executed in a subshell.

alias
alias name
alias name wordlist
The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name*; *wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias*.

break
Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while*. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw
Causes a break from a *switch*, resuming after the *endsw*.

case label:
A label in a *switch* statement as discussed below.

cd
cd name
chdir
chdir name
Change the shells working directory to directory *name*. If no argument is given, then change to the home directory of the user.

If *name* is not found as a subdirectory of the current directory (and does not begin with /, ./ or ../), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with /, then this is tried to see if it is a directory.

continue
Continue execution of the nearest enclosing *while* or *foreach*. The rest of the commands on the current line are executed.

default:
Labels the default case in a *switch* statement. The default should come after all *case* labels.

echo wordlist
echo **−n** wordlist
The specified words are written to the shells standard output, separated by spaces, and terminated with a newline unless the **−n** option is specified.

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

exec command
The specified command is executed in place of the current shell.

exit
exit(expr)
The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

foreach name (wordlist)
...
end
The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching *end* are executed. (Both *foreach* and *end* must appear alone on separate lines.)

The builtin command *continue* may be used to continue the loop prematurely and the builtin command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with ? before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal, you can rub it out.

glob wordlist
Like *echo* but no \ escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

goto word
The specified *word* is filename and command expanded to yield a string of the form "label". The shell rewinds its input as much as possible and searches for a line of the form "label:" possibly preceded by

blanks or tabs. Execution continues after the specified line.

history
Displays the history event list.

if (expr) command
If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is **not** executed (this is a bug).

if (expr) **then**
...
else if (expr2) **then**
...
else
...
endif
If the specified *expr* is true, then the commands to the first *else* are executed; else if *expr2* is true, then the commands to the second else are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The words *else* and *endif* must appear at the beginning of input lines; the *if* must appear alone on its input line or after an *else*.)

kill pid
kill −sig pid ...
Sends either the TERM (terminate) signal or specified signal to the specified processes. Signals are either given by number or names (as in **/usr/include/signal.h**, stripped of the prefix SIG). There is no default, saying "kill" does not send a signal to the current process. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal as well.

login
Terminate a login shell, replacing it with an instance of **/bin/login**. This is one way to log off, included for compatibility with *sh*(1).

logout
Terminate a login shell. Especially useful if *ignoreeof* is set.

newgrp
changes the group identification of its caller resulting in the access permissions being calculated with respect to the new group ID.

nice
nice +number
nice command
nice +number command
The first form sets the *nice* for this shell to 4. The second form sets the *nice* to the given number. The final two forms run command at priority 4 and *number* respectively. The super-user may specify negative niceness by using **nice** −**number** .... Command is always executed in a sub-shell, and the restrictions place on commands in simple *if* statements apply.

nohup
nohup command
The first form can be used in shell scripts to cause hangups to be

ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with & are effectively *nohup*ed.

onintr
onintr –
onintr label
Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form **onintr** – causes all interrupts to be ignored. The final form causes the shell to execute a **goto label** when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running detached and interrupts are being ignored, all forms of *onintr* have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

rehash
Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command
The specified *command* which is subject to the same restrictions as the *command* in the one line *if* statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

set
set name
set name=word
set name[index]=word
set name=(wordlist)
The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *indexth* component of name to word; this component must already exist. The final form sets *name* to the list of words in *wordlist*. In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note, however, that variable expansion happens for all arguments before any setting occurs.

setenv name value
Sets the value of environment variable *name* to be *value*, a single string. The variable PATH is automatically imported to and exported from the *csh* variable *path*; there is no need to use *setenv* for these.

shift
shift variable
The members of *argv* are shifted to the left, discarding *argv*[1]. It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified

variable.

source name
The shell reads commands from *name*. *Source* commands may be nested; if they are nested too deeply, the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands. Input during *source* commands is **never** placed on the history list.

switch (string)
case str1:
...
breaksw
...
default:
...
breaksw
endsw
Each case label is successively matched against the specified *string* which is first command and filename expanded. The file metacharacters *, ? and [...] may be used in the case labels, which are variable expanded. If none of the labels match before a "default" label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

time
time command
With no argument, a summary of time used by this shell and its children is printed. If arguments are given, the specified simple command is timed and a time summary as described under the *time* variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

umask
umask value
The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern
All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by **unalias***. It is not an error for nothing to be *unaliased*.

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

unset pattern
All variables whose names match the specified pattern are removed. Thus all variables are removed by **unset***; this has noticeably

distasteful side-effects. It is not an error for nothing to be *unset*.

wait
All background jobs are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (expr)
...
end
While the specified expression evaluates non-zero, the commands between the *while* and the matching end are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. (The *while* and *end* must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

@
@ name = expr
@ name[index] = expr
The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr*. If the expression contains <, >, & or |, then at least this part of the expression must be placed within ( ). The third form assigns the value of *expr* to the *index*th argument of *name*. Both *name* and its *index*th component must already exist.

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

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

Pre-defined and Environment Variables

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

This shell copies the environment variable USER into the variable *user*, TERM into *term*, and HOME into *home*, and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file ".cshrc" as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it.

argv Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e., "$1" is replaced by "$argv[1]", etc.

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

echo Set when the **−x** command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-builtin commands all expansions occur before

echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively.

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

home The home directory of the invoker, initialized from the environment. The filename expansion of "~" refers to this variable.

ignoreeof If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-Ds.

mail The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says "You have new mail." if the file exists with an access time not greater than its modify time.

If the first word of the value of *mail* is numeric, it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes.

If multiple mail files are specified, then the shell says "New mail in *name* when there is mail in the file *name*.

noclobber As described in the section on *Input/output*, restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that >> redirections refer to existing files.

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

nonomatch If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e., "echo [" still gives an error.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable, then only full path names will execute. The usual search path is ., **/bin** and **/usr/bin**, but this may vary from system to system. For the super-user the default search path is **/etc**, **/bin** and **/usr/bin**. A shell which is given neither the **-c** nor the **-t** option will normally hash the contents of the directories in the *path* variable after reading ".cshrc", and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the *rehash* or the commands may not be found.

prompt The string which is printed before each command is read from an interactive terminal input. If a ! appears in the string, it will be replaced by the current event number unless a preceding \ is given. Default is %, or # for the super-user.

shell The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of *Non-builtin Command Execution* below.) Initialized to the (system-dependent) home of the shell.

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status "1", all other builtin commands set status "0".

time Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.

verbose Set by the **−v** command line option, causes the words of each command to be printed after history substitution.

Non-builtin Command Execution

When a command to be executed is found not to be a builtin command, the shell attempts to execute the command via *exec*(2). Each word in the variable *path* names a directory from which the shell will attempt to execute the command. If it is given neither a **−c** nor a **−t** option, the shell will hash the names in these directories into an internal table so that it will only try an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash*), or if the shell was given a **−c** or **−t** argument, and in any case for each directory component of *path* which does not begin with a /, the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus **(cd ; pwd) ; pwd** prints the *home* directory; leaving you where you were (printing this after the home directory), while **cd ; pwd** leaves you in the *home* directory. Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands an a new shell is spawned to read it.

If there is an *alias* for *shell*, then the words of the alias will be prepended to the argument list to form the shell command. The first word of the *alias* should be the full path name of the shell (e.g., "$shell"). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

Argument List Processing

If argument 0 to the shell is **−**, then this is a login shell. The flag arguments are interpreted as follows:

-c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.

-e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

-f The shell will start faster, because it will neither search for nor execute commands from the file ".cshrc" in the invokers home directory.

-i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

-n Commands are parsed, but not executed. This may aid in syntactic checking of shell scripts.

-s Command input is taken from the standard input.

-t A single line of input is read and executed. A \ may be used to escape the newline at the end of this line and continue onto another line.

-v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.

-x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.

-V Causes the *verbose* variable to be set even before ".cshrc" is executed.

-X Is to **-x** as **-V** is to **-v**.

After processing of flag arguments, if arguments remain but none of the **-c**, **-i**, **-s**, or **-t** options was given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by "$0". Remaining arguments initialize the variable *argv*.

Signal Handling

The shell normally ignores *quit* signals. Processes running in background (by &) are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file ".logout".

EXAMPLE

csh

creates a new shell which will accept shell commands with Berkeley extensions.

FILES

~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by login shell, after ".cshrc" at login.
~/.logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a #.
/tmp/sh*	Temporary file for <<.

/etc/passwd Source of home directories for "~name".

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 5120 characters. The number of arguments to a command which involves filename expansion is limited to 1/6th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

SEE ALSO

sh(1), access(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), a.out(4), environ(4)
An Introduction to the C Shell, by William Joy.

BUGS

It suffices to place the sequence of commands in ()s to force it to a subshell, i.e., "(a ; b ; c)".

Control over tty output after processes are started is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by ?, are not placed in the *history* list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. All and more than one : modifier should be allowed on $ substitutions.

AUTHOR

William Joy.

NAME

csplit − context split

SYNOPSIS

csplit [−s] [−k] [−f prefix] file arg1 [... argn]

DESCRIPTION

Csplit reads *file* and separates it into n+1 sections, defined by the arguments *arg1... argn*. By default the sections are placed in xx00 ... xx*n* (*n* may not be greater than 99). These sections get the following pieces of *file*:

00: From the start of *file* up to (but not including) the line referenced by *arg1*.
01: From the line referenced by *arg1* up to the line referenced by *arg2*.
⋮
n+1: From the line referenced by *argn* to the end of *file*.

The options to *csplit* are:

−s *Csplit* normally prints the character counts for each file created. If the **−s** option is present, *csplit* suppresses the printing of all character counts.

−k *Csplit* normally removes created files if an error occurs. If the **−k** option is present, *csplit* leaves previously created files intact.

−f *prefix* If the **−f** option is used, the created files are named *prefix***00 ...** *prefixn*. The default is **xx00 ... xx***n*.

The arguments (*arg1 ... argn*) to *csplit* can be a combination of the following:

/rexp/ A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional + or − some number of lines (e.g., **/Page/−5**).

%rexp% This argument is the same as */rexp/*, except that no file is created for the section.

lnno A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.

{*num*} Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the Shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. *Csplit* does not affect the original file; it is the users responsibility to remove it.

EXAMPLE

```
csplit −f cobol file '/procedure division/' /par5./ /par16./
```

creates four files, "cobol00 ... cobol03". After editing the *split* files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

```
csplit -k file 100 {99}
```

splits the file at every 100 lines, up to 10,000 lines. The **-k** option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c '%main(%' '/^}/+1' {20}
```

assuming that "prog.c" follows the normal C coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in "prog.c".

SEE ALSO

ed(1), sh(1), regexp(5).

DIAGNOSTICS

Self explanatory except for:

```
arg - out of range
```

which means that the given argument did not reference a line between the current position and the end of the file.

NAME

ct − spawn getty to a remote terminal

SYNOPSIS

ct [**−h**] [**−v**] [**−wn**] [**−s**speed] telno ...

DESCRIPTION

Ct dials the phone number of a modem that is attached to a terminal, and spawns a *getty* process to that terminal. *Telno* is a telephone number, with equal signs for secondary dial tones and minus signs for delays at appropriate places. If more than one telephone number is specified, *ct* will try each in succession until one answers; this is useful for specifying alternate dialing paths.

Ct will try each line listed in the file **/usr/lib/uucp/L-devices** until it finds an available line with appropriate attributes or runs out of entries. If there are no free lines, *ct* will ask if it should wait for one, and if so, for how many minutes it should wait before it gives up. *Ct* will continue to try to open the dialers at one-minute intervals until the specified limit is exceeded. The dialogue may be overridden by specifying the **−w***n* option, where *n* is the maximum number of minutes that *ct* is to wait for a line.

Normally, *ct* will hang up the current line, so that that line can answer the incoming call. The **−h** option will prevent this action. If the **−v** option is used, *ct* will send a running narrative to the standard error output stream.

The data rate may be set with the **−s** option, where *speed* is expressed in baud. The default rate is 300.

After the user on the destination terminal logs out, *ct* prompts, **Reconnect?** If the response begins with the letter **n** the line will be dropped; otherwise, *getty* will be started again and the **login:** prompt will be printed.

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

EXAMPLE

```
ct -w15 -s1200 644-1234
```

dials from the terminal the given modem phone number (644-1234), spawning a login process at 1200 baud. If the dialer line is busy, *ct* will continue to try to open the dialer at one-minute intervals for a total of 15 minutes (as set by the **−w** option).

FILES

/usr/lib/uucp/L-devices
/usr/adm/ctlog

SEE ALSO

cu(1C), login(1), uucp(1C).

NAME

ctags − maintain a tags file for a C program

SYNOPSIS

ctags [**−a**] [**−u**] [**−w**] [**−x**] name ...

DESCRIPTION

Ctags makes a tags file for *ex*(1) and *vi*(1) from the specified C, Fortran, and Pascal sources.

A tags file gives the locations of specified objects (in this case functions) in a group of files. Each line of the tags file contains the function name, the file in which it is defined, and a scanning pattern used to find the function definition. These are given in separate fields on the line, separated by blanks or tabs. Using the *tags* file, *ex* can quickly find these function definitions.

Options

The **−a** option causes the output to be appended to the tags file instead of rewriting it.

The **−u** option causes the specified files to be *updated* in tags, that is, all references to them are deleted, and the new values are appended to the file. This option implies the **−a** option. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.)

The **−w** option suppresses warning diagnostics.

If the **−x** flag is given, *ctags* produces a list of function names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output.

Files whose name ends in ".c" or ".h" are assumed to be C source files and are searched for C routine and macro definitions.

The tag *main* is treated specially in C programs. The tag formed is created by prepending "M" to the name of the file, with a trailing ".c" removed, if any, and leading pathname components also removed. This makes use of *ctags*, practical in directories with more than one program.

EXAMPLE

```
ctags *.c *.h
```

puts the tags from all the ".c" and ".h" files into the tagsfile "tags".

FILES

tags output tags file

SEE ALSO

ex(1), vi(1).

BUGS

Not all warning diagnostics are suppressed by **−w**.

AUTHOR

Ken Arnold

NAME

cu − call another UNIX System

SYNOPSIS

cu [−sspeed] [−lline] [−**h**] [−**t**] [−**d**] [−**m**] [−**o**|−**e**] telno | **dir**

DESCRIPTION

Cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of ASCII files. *Speed* gives the transmission speed (110, 150, 300, 600, 1200, 4800, 9600); 300 is the default value. Most of our modems are either 300 or 1200 baud. For dial out lines, *cu* will choose a modem speed (300 or 1200) as the slowest available which will handle the specified transmission speed. Directly connected lines may be set to speeds higher than 1200 baud.

The −**l** value may be used to specify a device name for the communications line device to be used. This can be used to override searching for the first available line having the right speed. The speed of a line is taken from the file **/usr/lib/uucp/L-devices**, overriding any speed specified by the −**s** option. The −**h** option emulates local echo, supporting calls to other computer systems which expect terminals to be in half-duplex mode. The −**t** option is used when dialing an ASCII terminal which has been set to auto-answer. Appropriate mapping of carriage-returns to carriage-return-line-feed pairs is set. The −**d** option cause diagnostic traces to be printed. The −**m** option specifies a direct line which has modem control. The −**e** (−**o**) option designates that even (odd) parity is to be generated for data sent to the remote. *Telno* is the telephone number, with equal signs for secondary dial tone or minus signs for delays, at appropriate places. The string **dir** for *telno* may be used for directly connected lines, and implies a null ACU. Using **dir** insures that a line has been specified by the −**l** option.

Cu will try each line listed in the file **/usr/lib/uucp/L-devices** until it finds an available line with appropriate attributes or runs out of entries. After making the connection, *cu* runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~ , passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with ~ , passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following:

~.	terminate the conversation.
~!	escape to an interactive shell on the local system.
~!*cmd...*	run *cmd* on the local system (via **sh** −**c**).
~$*cmd...*	run *cmd* locally and send its output to the remote system.
~%**take** *from* [*to*]	copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
~%**put** *from* [*to*]	copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places.

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

~%nostop turn off the DC3/DC1 input control protocol for the remainder of the session. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters,

The *receive* process normally copies data from the remote system to its standard output. A line from the remote that begins with ~> initiates an output diversion to a file. The complete sequence is:

~> [>]: *file*
zero or more lines to be written to *file*
~>

Data from the remote is diverted (or appended, if >> is used) to file. The trailing ~> terminates the diversion.

The use of **~%put** requires *stty*(1) and *cat*(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of **~%take** requires the existence of *echo*(1) and *cat*(1) on the remote system. Also, **stty tabs** mode should be set on the remote system if tabs are to be copied without expansion.

EXAMPLE

cu -s 1200 777-8888

attempts to connect to the telephone line numbered "777-8888" at 1200 baud rate.

FILES

/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..(tty-device)
/dev/null

SEE ALSO

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

DIAGNOSTICS

Exit code is zero for normal exit, non-zero (various values) otherwise.

BUGS

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

NAME

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

SYNOPSIS

cut −**c**list [file1 file2 ...]
cut −**f**list [−**d**char] [−**s**] [file1 file2 ...]

DESCRIPTION

Use *cut* to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, i.e., character positions as on a punched card (−**c** option), or the length can vary from line to line and be marked with a field delimiter character like *tab* (−**f** option). *Cut* can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

list A comma-separated list of integer field numbers (in increasing order), with optional − to indicate ranges as in the −**o** option of *nroff*/*troff* for page ranges; e.g., **1,4,7**; **1−3,8**; **−5,10** (short for **1−5,10**); or **3−** (short for third through last field).

−**c** *list* The *list* following −**c** (no space) specifies character positions (e.g., −**c1−72** would pass the first 72 characters of each line).

−**f** *list* The *list* following −**f** is a list of fields assumed to be separated in the file by a delimiter character (see −**d**); e.g. , −**f1,7** copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless −**s** is specified.

−**d** *char* The character following −**d** is the field delimiter (−**f** option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.

−**s** Suppresses lines with no delimiter characters in case of −**f** option. Unless specified, lines with no delimiters will be passed through untouched.

Either the −**c** or −**f** option must be specified.

HINTS

Use *grep*(1) to make horizontal "cuts" (by context) through a file, or *paste*(1) to put files together column-wise (i.e., horizontally). To reorder columns in a table, use *cut* and *paste*.

EXAMPLE

```
cut −d: −f1,5 /etc/passwd
```

mapping of user IDs to names.

```
name=`who am i | cut −f1 −d" "`
```

to set *name* to current login name.

DIAGNOSTICS

line too long
A line can have no more than 511 characters or fields.

bad list for c/f option
Missing −**c** or −**f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

no fields
The *list* is empty.

SEE ALSO
grep(1), paste(1).

NAME

cw, checkcw − prepare constant-width text for troff

SYNOPSIS

cw [**-lxx**] [**-rxx**] [**-fn**] [**-t**] [**+t**] [**-d**] [files]

checkcw [**-lxx**] [**-rxx**] files

DESCRIPTION

Cw is a preprocessor for *troff*(1) input files that contain text to be typeset in the constant-width (CW) font.

Text typeset with the CW font resembles the output of terminals and of line printers. This font is used to typeset examples of programs and of computer output in user manuals, programming texts, etc. (An earlier version of this font was used in typesetting *The C Programming Language* by B. W. Kernighan and D. M. Ritchie.) It has been designed to be quite distinctive (but not overly obtrusive) when used together with the Times Roman font.

Because the CW font contains a "non-standard" set of characters and because text typeset with it requires different character and inter-word spacing than is used for "standard" fonts, documents that use the CW font must be preprocessed by *cw*.

The CW font contains the 94 printing ASCII characters:

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!$%&()'"*+@.,/:;=?[]|-_^~"<>{}#
```

plus eight non-ASCII characters represented by four-character *troff*(1) names (in some cases attaching these names to "non-standard" graphics):

Character	*Symbol*	*Troff Name*
"Cents" sign	¢	\(ct
EBCDIC "not" sign	¬	\(no
Left arrow	←	\(<-
Right arrow	→	\(->
Down arrow	↓	\(da
Vertical single quote	'	\(fm
Control-shift indicator	†	\(dg
Visible space indicator	□	\(sq
Hyphen	-	\(hy

The hyphen is a synonym for the unadorned minus sign (-). Certain versions of *cw* recognize two additional names: \(ua for an up arrow and \(lh for a diagonal left-up (home) arrow.

Cw recognizes five request lines, as well as user-defined delimiters. The request lines look like *troff*(1) macro requests, and are copied in their entirety by *cw* onto its output; thus, they can be defined *by the user* as *troff*(1) macros; in fact, the .CW and .CN macros *should* be so defined (see *HINTS* below). The five requests are:

.CW Start of text to be set in the CW font; .CW causes a break; it can take precisely the same options, in precisely the same format, as are available on the *cw* command line.

.CN End of text to be set in the CW font; .CN causes a break; it can take the same options as are available on the *cw* command line.

.CD Change delimiters and/or settings of other options; takes the same options as are available on the *cw* command line.

.CP *arg1 arg2 arg3 ... argn*
All the arguments (which are delimited like *troff*(1) macro arguments) are concatenated, with the odd-numbered arguments set in the CW font and the even-numbered ones in the prevailing font.

.PC *arg1 arg2 arg3 ... argn*
Same as .CP, except that the even-numbered arguments are set in the CW font and the odd-numbered ones in the prevailing font.

The .CW and .CN requests are meant to bracket text (e.g., a program fragment) that is to be typeset in the CW font "as is." Normally, *cw* operates in the *transparent* mode. In that mode, except for the .CD request and the nine special four-character names listed in the table above, every character between .CW and .CN request lines stands for itself. In particular, *cw* arranges for periods (.) and apostrophes (') at the beginning of lines, and backslashes (\) everywhere to be "hidden" from *troff*(1). The transparent mode can be turned off (see below), in which case normal *troff*(1) rules apply; in particular, lines that begin with . and ' are passed through untouched (except if they contain delimiters—see below). In either case, *cw* hides the effect of the font changes generated by the .CW and .CN requests; *cw* also defeats all ligatures (*fi*, *ff*, etc.) in the CW font.

The only purpose of the .CD request is to allow the changing of various options other than just at the beginning of a document.

The user can also define *delimiters*. The left and right delimiters perform the same function as the .CW/.CN requests; they are meant, however, to enclose CW "words" or "phrases" in running text (see example under *BUGS* below). *Cw* treats text between delimiters in the same manner as text enclosed by .CW/.CN pairs, except that, for aesthetic reasons, spaces and backspaces inside .CW/.CN pairs have the same width as other CW characters, while spaces and backspaces between delimiters are half as wide, so they have the same width as spaces in the prevailing text (but are *not* adjustable). Font changes due to delimiters are *not* hidden.

Delimiters have no special meaning inside .CW/.CN pairs.

The options are:

−l*xx* The one- or two-character string *xx* becomes the left delimiter; if *xx* is omitted, the left delimiter becomes undefined, which it is initially.

−r*xx* Same for the right delimiter. The left and right delimiters may (but need not) be different.

−f*n* The CW font is mounted in font position *n*; acceptable values for *n* are 1, 2, and 3 (default is 3, replacing the bold font). This option is only useful at the beginning of a document.

−t Turn transparent mode *off*.

+t Turn transparent mode *on* (this is the initial default).

−d Print current option settings on file descriptor 2 in the form of *troff*(1) comment lines. This option is meant for debugging.

Cw reads the standard input when no *files* are specified (or when - is specified as the last argument), so it can be used as a filter. Typical usage is:

cw *files* | troff ...

Checkcw checks that left and right delimiters, as well as the .CW/.CN pairs, are properly balanced. It prints out all offending lines.

HINTS

Typical definitions of the ..CW and ..CN macros meant to be used with the *mm*(7) macro package:

```
.de CW
.DS I
.ps 9
.vs 10.5p
.ta 16m/3u 32m/3u 48m/3u 64m/3u 80m/3u 96m/3u ...
..
.de CN
.ta 0.5i 1i 1.5i 2i 2.5i 3i 3.5i 4i 4.5i 5i 5.5i 6i
.vs
.ps
.DE
..
```

At the very least, the .CW macro should invoke the *troff*(1) no-fill (.nf) mode.

When set in running text, the CW font is meant to be set in the same point size as the rest of the text. In displayed matter, on the other hand, it can often be profitably set one point *smaller* than the prevailing point size (the displayed definitions of .CW and .CN above are one point smaller than the running text on this page). The CW font is sized so that, when it is set in 9-point, there are 12 characters per inch.

Documents that contain CW text may also contain tables and/or equations. If this is the case, the order of preprocessing should be: *cw*, *tbl*, and *eqn*. Usually, the tables contained in such documents will not contain any CW text, although it is entirely possible to have *elements* of the table set in the CW font; of course, care must be taken that *tbl*(1) format information not be modified by *cw*. Attempts to set equations in the CW font are not likely to be either pleasing or successful.

In the CW font, overstriking is most easily accomplished with backspaces: letting ← represent a backspace, d←←† yields %‡% . Because spaces (and, therefore backspaces) are half as wide between delimiters as inside .CW/.CN pairs (see above), two backspaces are required for each overstrike between delimiters.

EXAMPLE

cw text | tbl | troff -mm

processes the text file "text", sends the output to *tbl*(1) and then sends the output for final formatting to *troff*(1) and *mm*(7).

FILES

/usr/lib/font/ftCW CW font-width table

SEE ALSO

eqn(1), mmt(1), tbl(1), troff(1), mm(5), mv(5).

WARNINGS

If text preprocessed by *cw* is to make any sense, it must be set on a typesetter equipped with the CW font or on a STARE facility; on the latter, the CW font appears as bold, but with the proper CW spacing.

BUGS

Only a masochist would use periods (.), backslashes (\), or double quotes (") as delimiters, or as arguments to .CP and .PC.
Certain CW characters don't concatenate gracefully with certain Times Roman characters, e.g., a CW ampersand (&) followed by a Times Roman comma(,); in such cases, judicious use of *troff*(1) half- and quarter-spaces (\| and \^) is most salutary, e.g., one should use _&_\^, (rather than just plain _&_,) to obtain &, (assuming that _ is used for both delimiters).
Using *cw* with *nroff* is silly.
The output of *cw* is hard to read.
See also *BUGS* under *troff*(1).

NAME

cxref − generate C program cross reference

SYNOPSIS

cxref [options] files

DESCRIPTION

Cxref analyzes a collection of C files and attempts to build a cross reference table. *Cxref* utilizes a special version of *cpp* to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the **−c** option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

In addition to the **−D**, **−I** and **−U** options (which are identical to their interpretation by *cc*(1)), the following *options* are interpreted by *cxref*:

−c Print a combined cross-reference of all input files.

−w<num>
Width option which formats output no wider than <num> (decimal) columns. This option will default to 80 if <num> is not specified or is less than 51.

−o *file* Direct output to named *file.*

−s Operate silently; does not print input file names.

−t Format listing for 80-column width.

FILES

/usr/lib/xcpp special version of C-preprocessor.

SEE ALSO

cc(1).

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you can't compile these files, anyway.

NAME

date − print and set the date

SYNOPSIS

date [mmddhhmm[yy]] [+format]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight time.

If the argument begins with +, the output of *date* is under the control of the user. The format for the output is similar to that of the first argument to *printf*(3S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year − 01 to 12
d	day of month − 01 to 31
y	last 2 digits of year − 00 to 99
D	date as mm/dd/yy
H	hour − 00 to 23
M	minute − 00 to 59
S	second − 00 to 59
T	time as HH:MM:SS
j	day of year − 001 to 366
w	day of week − Sunday = 0
a	abbreviated weekday − Sun to Sat
h	abbreviated month − Jan to Dec
r	time in AM/PM notation

EXAMPLE

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

generates as output:

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

DIAGNOSTICS

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

WARNING

It is a bad practice to change the date while the system is running multi-user.

NAME

dc — desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number

The value of the number is pushed on the stack. A number is an unbroken string of the digits 0–9. It may be preceded by an underscore (_) to input a negative number. Numbers may contain decimal points.

+ − / * % ^

The top two values on the stack are added (+), subtracted (−), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

s*x* The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the s is capitalized, *x* is treated as a stack and the value is pushed on it.

l*x* The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the l is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d The top value on the stack is duplicated.

p The top value on the stack is printed. The top value remains unchanged. **P** interprets the top of the stack as an ASCII string, removes it, and prints it.

f All values on the stack are printed.

q exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped and the string execution level is popped by that value. Alternately, control-d (EOF) will exit from *dc*.

x treats the top element of the stack as a character string and executes it as a string of *dc* commands.

X replaces the number on the top of the stack with its scale factor.

[...] puts the bracketed ASCII string onto the top of the stack.

<*x* >*x* =*x*

The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.

v replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

!	interprets the rest of the line as a UNIX System command.
c	All values on the stack are popped.
i	The top value on the stack is popped and used as the number radix for further input. I pushes the input base on the top of the stack.
o	The top value on the stack is popped and used as the number radix for further output.
O	pushes the output base on the top of the stack.
k	the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
z	The stack level is pushed onto the stack.
Z	replaces the number on the top of the stack with its length.
?	A line of input is taken from the input source (usually the terminal) and executed.
; :	are used by *bc* for array operations.

EXAMPLE

```
dc
24.2 56.2 + p
```

adds the two numbers and prints the result (top value in the stack).

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

prints the first ten values of *n*!.

SEE ALSO

bc(1), which is a preprocessor for *dc* providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

DIAGNOSTICS

x is unimplemented	where *x* is an octal number.
stack empty	for not enough elements on the stack to do what was asked.
Out of space	when the free list is exhausted (too many digits).
Out of headers	for too many numbers being kept around.
Out of pushdown	for too many items on the stack.
Nesting Depth	for too many levels of nested execution.

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

option	*values*
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input records before starting copy
seek=*n*	seek *n* records from beginning of output file before copying; *dd* creates the specified output file (see *creat*(2)), which insures the length of the file will be zero; seeking *n* records from the beginning of the output file will fill the skipped area with zeros (nulls).
count=*n*	copy only *n* input records
conv=**ascii**	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
..., ...	several comma-separated conversions
multi=**in**	input file is multi-volume
out	output file is multi-volume
in,out	both the input file and the output file are multi-volume

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if *ascii*, *ebcdic*, or *ibm* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and new-line added before sending the line to the output. In the latter two cases ASCII characters are read into the conversion buffer, converted to EBCDIC (or the IBM version of EBCDIC), and blanks added to make up an output record of size *cbs*.

If multi-volume input(output) is specified, a prompt is given on end-of-file to allow another volume to be mounted.

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

EXAMPLE

```
dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase
```

will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file "x".

Note the use of raw magtape. *Dd* is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

SEE ALSO

cp(1).

DIAGNOSTICS

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

BUGS

The ASCII/ EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The *ibm* conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

New-lines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

NAME

delta − make a delta (change) to an SCCS file

SYNOPSIS

delta [−rSID] [−s] [−n] [−glist] [−m[mrlist]] [−y[comment]] [−p] files

DESCRIPTION

Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

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

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin*(1)) that may be present in the SCCS file (see **−m** and **−y** keyletters below).

Keyletter arguments apply independently to each named file.

−r*SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding *get*s for editing (**get −e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **−r** keyletter can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

−s Suppresses the issue on the standard output of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

−n Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

−g*list* Specifies a *list* (see *get*(1) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (SID) created by this delta.

−m[*mrlist*] If the SCCS file has the **v** flag set (see *admin*(1)) then a Modification Request (**MR**) number *must* be supplied as the reason for creating the new delta.

If **−m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **−y** keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MR** list.

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the **MR** numbers. If a non-zero exit status is returned from **MR** number validation program, *delta* terminates (it is assumed that the **MR** numbers were not all valid).

−y[*comment*] Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment.*

If **−y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

−p Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

EXAMPLE

```
% delta s.test1.c
comments? second version
1.2
1 inserted
0 deleted
12 unchanged
```

does a *delta* on file "test1.c".

FILES

All files of the form *?*-file are explained in the *Source Code Control System User's Guide.* The naming convention for these files is also described there.

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the *g-file.*

WARNINGS

Lines beginning with an **SOH** ASCII character (binary 001) cannot be placed in the SCCS file unless the **SOH** is escaped. This character has special meaning to SCCS (see *sccsfile*(5)) and will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get/delta* sequences should be used.

If the standard input (−) is specified on the *delta* command line, the **−m** (if necessary) and −y keyletters *must* also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.

SEE ALSO

admin(1), bdiff(1), cdc(1), get(1), help(1), prs(1), rmdel(1), sccsfile(4).
Source Code Control System User's Guide

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

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

SYNOPSIS

deroff [−**mx**] [−**w**] [files]

DESCRIPTION

Deroff reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between **.EQ** and **.EN** lines, and between delimiters), and *tbl*(1) descriptions, perhaps replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. *Deroff* follows chains of included files (**.so** and **.nx** *troff* commands); if a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

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

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

EXAMPLE

```
deroff textfile
```

removes all *nroff*, *troff*, and macro definitions from "textfile".

SEE ALSO

eqn(1), nroff(1), tbl(1), troff(1).

BUGS

Deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.

The −**ml** option does not handle nested lists correctly.

NAME

diff − differential file comparator

SYNOPSIS

diff [**−efbh**] file1 file2

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is −, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

n1 **a** *n3,n4*
n1,n2 **d** *n3*
n1,n2 **c** *n3,n4*

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs where *n1* = *n2* or *n3* = *n4* are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

The **−b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **−e** option produces a script of *a*, *c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **−f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **−e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed − $1
```

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option **−h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **−e** and **−f** are unavailable with **−h**.

EXAMPLE

```
diff -e file1 file2
```

where "file1" and "file2" are two versions of the manual text for the *cp* command, produces:

```
35,41d
27c
In the second form, one or more

18,25c
existed; the mode of the source file
is used otherwise.
```

```
15c
The mode and owner of

10c
file ... directory

7c
file1 file2

1,3c
.TH CP 1
.SH NAME
```

Following this *ed* script would transform "file1" into file2", line for line and character for character.

FILES

/tmp/d?????
/usr/lib/diffh for **−h**

SEE ALSO

cmp(1), comm(1), ed(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

BUGS

Editing scripts produced under the **−e** or **−f** option are naive about creating lines consisting of a single period (.).

NAME

diff3 − 3-way differential file comparison

SYNOPSIS

diff3 [−**ex3**] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	*file1* is different
====2	*file2* is different
====3	*file3* is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

f: *n1* **a** Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.

f: *n1* , *n2* **c** Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the −**e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged ==== and ====3. Option −**x** (−**3**) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to *file1*.

```
(cat script; echo '1,$p') | ed − file1
```

EXAMPLE

If file "f1" contains the following text:

```
This is a file.
This is the first of three files.
This is not the last file.
```

and file "f2" contains:

```
This is a file.
This is the second of three files.
This is not the last file.
```

and file "f3" contains:

```
This is a file.
This is the third of three files.
This is the last file.
```

then

```
diff3 f1 f2 f3
```

will return

```
====
1:2,3c
     This is the first of three files.
     This is not the last file.
```

```
2:2,3c
     This is the second of three files.
     This is not the last file.
3:2,3c
     This is the third of three files.
     This is the last file
```

FILES

/tmp/d3*
/usr/lib/diff3prog

SEE ALSO

diff(1).

BUGS

Text lines that consist of a single . will defeat −e.
Files longer than 64K bytes won't work.

NAME

diffdir − diff directories

SYNOPSIS

diffdir [**−h**] [**−s**] dir1 dir2

DESCRIPTION

Diffdir compares the differences of two directories recursively by sorting the contents of the directories by name and then runs a *diff* on text files which are different. Object files which differ and files which appear in only one directory are also listed.

The **−h** option causes *diffdir* to paginate its output, and to summarize binary differences and files in only one place at the end of the *diff*. Each individual *diff* is run through an appropriate *pr*.

The **−s** option causes files which are the same to be reported; normally they are omitted.

EXAMPLE

```
diffdir dir1 dir2
```

compares all the files in two directories and reports differences, by line number, for similar files. Unique files are simply listed.

FILES

/usr/bin/cmp compare two files

SEE ALSO

diff(1).

BUGS

Program should pass flags through to *diff*.

AUTHOR

Bill Joy

NAME

diffmk − mark differences between files

SYNOPSIS

diffmk name1 name2 name3

DESCRIPTION

Diffmk compares two versions of a file and creates a third file that includes "change mark" commands for *nroff*(1) or *troff*(1). *Name1* and *name2* are the old and new versions of the file. *Diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter "change mark" (**.mc**) requests. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

If the characters | and * are inappropriate, a copy of *diffmk* can be edited to change them (*diffmk* is a shell procedure).

If anyone is so inclined, *diffmk* can be used to produce listings of C (or other) programs with changes marked.

EXAMPLE

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

produces a listing of two versions of a C program with changes marked. First the two versions are compared and a new file, "tmp", is created containing the *change mark* commands. The temporary file is then passed to *nroff*(1) using the file "macs" which contains:

```
.pl 1
.ll 77
.nf
.eo
.nc `
```

The *.ll* request might specify a different line length, depending on the nature of the program being printed. The *.eo* and *.nc* requests are probably needed only for C programs.

SEE ALSO

diff(1), nroff(1), troff(1).

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing **.sp** by **.sp 2** will produce a "change mark" on the preceding or following line of output.

NAME

dircmp − directory comparison

SYNOPSIS

dircmp [**−d**] [**−s**] dir1 dir2

DESCRIPTION

Dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the filenames common to both directories have the same contents.

−d Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff*(1).

−s Suppress messages about identical files.

EXAMPLE

```
dircmp d1 d2
```

will show the differences between the directories **d1** and **d2**.

SEE ALSO

cmp(1), diff(1).

NAME

du – summarize disk usage

SYNOPSIS

du [**–ars**] [names]

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, . is used.

The optional argument **–s** causes only the grand total (for each of the specified *names*) to be given. The optional argument **–a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

Du is normally silent about directories that cannot be read, files that cannot be opened, etc. The **–r** option will cause *du* to generate messages in such instances.

A file with two or more links is only counted once.

EXAMPLE

```
du dir1 dir2
```

produces a count of the number of blocks in each of the directories. In order to see how many blocks are in each file, the **–a** option must be used.

BUGS

If the **–a** option is not used, non-directories given as arguments are not listed.

If there are too many distinct linked files, *du* will count the excess files more than once.

Files with holes in them will get an incorrect block count.

NAME

echo – echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

- **\b** backspace
- **\c** print line without new-line
- **\f** form-feed
- **\n** new-line
- **\r** carriage return
- **\t** tab
- **** backslash
- *****n* the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number *n*, which must start with a zero.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

EXAMPLE

```
echo curmudgeon
```

simply responds

```
curmudgeon
```

on the standard output.

SEE ALSO

sh(1).

NAME

ed, red − text editor

SYNOPSIS

ed [−] [−**x**] [file]

red [−] [−**x**] [file]

DESCRIPTION

Ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited. The optional − suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a !*shell command*. If −**x** is present, an *x* command is simulated first to handle an encrypted file. *Ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

Red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via !*shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec*(4) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty −tabs** or **stty tab3** mode (see *stty*(1), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10 and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

Ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character REs* match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]; see 1.4 below).

b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).

c. $ (currency symbol), which is special at the *end* of an entire RE (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)

1.3 A period (.) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (-) may be used to indicate a range of consecutive ASCII characters; for example, **[0-9]** is equivalent to **[0123456789]**. The - loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., **[]a-f]** matches either a right square bracket (]) or one of the letters **a** through **f** inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *RE*s from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly* *m* occurrences; \{*m*,\} matches *at least* *m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between* *m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the

same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression **^\(.*\)\1$** matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both):

3.1 A circumflex (**^**) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

3.2 A currency symbol (**$**) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction **^** *entire RE***$** constrains the entire RE to match the entire line.

The null RE (e.g., //) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character **.** addresses the current line.
2. The character **$** addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. *'x* addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before *FILES* below.
6. A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign (+) or a minus sign (−) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or −, the addition or subtraction is taken with respect to the current line; e.g, −5 is understood to mean .−5.
9. If an address ends with + or −, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address − refers to the line preceding the current line. (To maintain compatibility with earlier versions of the

editor, the character ^ in addresses is entirely equivalent to −.) Moreover, trailing + and − characters have a cumulative effect, so −− refers to the current line less 2.

10. For convenience, a comma (,) stands for the address pair **1,$**, while a semicolon (;) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6. above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n** or **p**, in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n* and *p* commands.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the newline character).

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file* The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell

command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E *file* The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file* If *file* is given, the *f*ile-name command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1,$)g/*RE/command list*

In the *g*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted; the . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before *FILES* below.

(1,$)G/*RE/*

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an **&** causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h The *h*elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent **?** diagnostics. It will also explain the previous **?** if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the newline character).

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given,

this command does nothing.

(.)k*x* The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. The address *'x* then addresses this line; . is unchanged.

(.,.)l The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab, backspace*) are represented by (hopefully) mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)m*a* The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file; it is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(.,.)n The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p The *p*rint command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*; for example, *dp* deletes the current line and prints the new current line.

P The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q The *q*uit command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

Q The editor exits without checking if changes have been made in the buffer since the last *w* command.

($)r *file* The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/ *RE*/ *replacement*/ or
(.,.)s/ *RE*/ *replacement*/**g**
The *s*ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first

occurrence of the matched string is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a* This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1,$)v/*RE/command list*

This command is the same as the global command *g* except that the *command list* is executed with . initially set to every line that does *not* match the RE.

(1,$)V/*RE/*

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the RE.

(1,$)w *file*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

X A key string is demanded from the standard input. Subsequent *e*, *r*, and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

(.)= The line number of the addressed line is typed; **.** is unchanged by this command.

!*shell command*
The remainder of the line after the ! is sent to the UNIX System shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character **%** is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, **!!** will repeat the last shell command. If any expansion is performed, the expanded line is echoed; **.** is unchanged.

(.+1)<new-line>
An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a ? and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If the closing delimiter of a RE or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

EXAMPLE

ed text

would invoke the editor with the file named "text". For further examples, see "*A Tutorial Introduction to the UNIX Text Editor*" and "*Advanced Editing on UNIX*"

FILES

/tmp/e#	temporary; # is the process number.
ed.hup	work is saved here if the terminal is hung up.

DIAGNOSTICS

?	for command errors.
?*file*	for an inaccessible file.
	(use the *h*elp and *H*elp commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy

ed's buffer via the *e* or *q* commands: it prints ? and allows one to continue editing. A second *e* or *q* command at this point will take effect. The − command-line option inhibits this feature.

SEE ALSO

crypt(1), grep(1), sed(1), sh(1), stty(1), fspec(4), regexp(5).
A Tutorial Introduction to the UNIX Text Editor, by B. W. Kernighan.
Advanced Editing on UNIX , by B. W. Kernighan.

CAVEATS AND BUGS

A *!* command cannot be subject to a *g* or a *v* command.
The *!* command and the ! escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).
The sequence **\n** in a RE does not match a new-line character.
The *l* command mishandles DEL.
Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.
Characters are masked to 7 bits on input.

NOTE

The **−x** option and the editor command **X** are not implemented in the international distribution.

NAME

efl − Extended Fortran Language

SYNOPSIS

efl [options] [files]

DESCRIPTION

Efl compiles a program written in the EFL language into clean Fortran on the standard output. *Efl* provides the C-like control constructs similar to *ratfor*:

statement grouping with braces.

decision-making:
if, **if-else**, and **select-case** (also known as **switch-case**);
while, **for**, Fortran **do**, **repeat**, and **repeat** ... **until** loops;
multi-level **break** and **next**.

EFL has C-like data structures, e.g.:

```
struct
    {
    integer flags(3)
    character(8) name
    long real coords(2)
    } table(100)
```

The language offers generic functions, assignment operators (+=, &=, etc.), and sequentially evaluated logical operators (**&&** and **||**). There is a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) })
```

EFL also provides some syntactic "sugar":

free-form input:
multiple statements per line; automatic continuation; statement label names (not just numbers).

comments:
this is a comment.

translation of relational and logical operators:
>, **>=**, **&**, etc., become **.GT.**, **.GE.**, **.AND.**, etc.

return expression to caller from function:
return (*expression*)

defines:
define *name replacement*

includes:
include *file*

Efl understands several option arguments: **−w** suppresses warning messages, **−#** suppresses comments in the generated program, and the default option **−C** causes comments to be included in the generated program.

An argument with an embedded = (equal sign) sets an EFL option as if it had appeared in an **option** statement at the start of the program. Many options are described in the reference manual. A set of defaults for a particular target machine may be selected by one of the choices: **system=unix**, **system=gcos**, or **system=cray**. The default setting of the

system option is the same as the machine the compiler is running on. Other specific options determine the style of input/output, error handling, continuation conventions, the number of characters packed per word, and default formats.

Efl is best used with *fortran*(1).

EXAMPLE

```
efl prog.for | fortran -o prog
```

will process the program *prog.for* through *efl* and then run the *fortran*(1) compiler on the output from *efl*, generating an executable file named "prog".

SEE ALSO

cc(1), fortran(1).
The Programming Language EFL by S.I. Feldman.

NAME

enable, disable − enable/disable LP printers

SYNOPSIS

enable printers
disable [−**c**] [−**r**[reason]] printers

DESCRIPTION

Enable activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

Disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

−**c** Cancel any requests that are currently printing on any of the designated printers.

−**r**[*reason*] Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next −**r** option. If the −**r** option is not present or the −**r** option is given without a reason, then a default reason will be used. *Reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).

NAME

env − set environment for command execution

SYNOPSIS

env [−] [name=value] ... [command args]

DESCRIPTION

Env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The − flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

EXAMPLE

```
env XYZ=pdq sh
```

sets the environment name "XYZ" to the value *pdq* for the duration of the new shell.

SEE ALSO

sh(1), exec(2), profile(4), environ(5).

NAME

eqn, neqn, checkeq − format mathematical text for nroff or troff

SYNOPSIS

eqn [−**d**xy] [−**p**n] [−**s**n] [−**f**n] [files]

neqn [−**d**xy] [−**p**n] [−**s**n] [−**f**n] [files]

checkeq [files]

DESCRIPTION

Eqn is a *troff*(1) preprocessor for typesetting mathematical text on a phototypesetter, while *neqn* is used for the same purpose with *nroff* on typewriter-like terminals. Usage is almost always:

```
eqn files | troff
neqn files | nroff
```

or equivalent.

If no files are specified (or if − is specified as the last argument), these programs read the standard input. A line beginning with **.EQ** marks the start of an equation; the end of an equation is marked by a line beginning with **.EN**. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to designate two characters as *delimiters*; subsequent text between delimiters is then treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument −**d***xy* or (more commonly) with **delim** *xy* between **.EQ** and **.EN**. The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by **delim off**. All text that is neither between delimiters nor between **.EQ** and **.EN** is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and **.EQ**/**.EN** pairs.

Tokens within *eqn* are separated by spaces, tabs, new-lines, braces, double quotes, tildes, and circumflexes. Braces { } are used for grouping; generally speaking, anywhere a single character such as *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full space in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub j* makes x_j, *a sub k sup 2* produces a_k^2, while $e^{x^2+y^2}$ is made with *e sup {x sup 2 + y sup 2}*. Fractions are made with **over**: *a over b* yields $\frac{a}{b}$; **sqrt** makes square roots: *1 over sqrt {ax sup 2+bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits: $\lim_{n\to\infty}\sum_{0}^{n} x_i$ is made with *lim from {n −> inf} sum from 0 to n x sub i*. Left and right brackets, braces, etc., of the right height are made with **left** and **right**: *left [x sup 2 + y sup 2 over alpha right] ~ =~ 1* produces $\left[x^2+\frac{y^2}{\alpha}\right] = 1$.

Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket). A **left** *thing* need not have a matching **right** *thing*.

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$. Piles may have arbitrary numbers of elements; **lpile** left-justifies, **pile** and **cpile** center (but with different vertical spacing), and **rpile** right justifies. Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x} = \overline{f(t)}$, *y dotdot bar ~ =~ n under* is $\ddot{\bar{y}} = \underline{n}$, and *x vec ~ =~ y dyad* is $\vec{x} = \vec{y}$.

Point sizes and fonts can be changed with **size** *n* or **size** ± *n*, **roman**, **italic**, **bold**, and **font** *n*. Point sizes and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **−s***n* and **−f***n*.

Normally, subscripts and superscripts are reduced by 3 points from the previous size; this may be changed by the command-line argument **−p***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

```
define thing % replacement %
```

defines a new token called *thing* that will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords such as **sum** (∑), **int** (∫), **inf** (∞), and shorthands such as >= (≥), != (≠), and −> (→) are recognized. Greek letters are spelled out in the desired case, as in **alpha** (α), or **GAMMA** (Γ). Mathematical words such as **sin**, **cos**, and **log** are made Roman automatically. *Troff*(1) four-character escapes such as **\(dd** (‡) and **\(bs** (⧵) may be used anywhere. Strings enclosed in double quotes ("...") are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff*(1) when all else fails. Full details are given in the manual cited below.

EXAMPLE

```
eqn file1 | troff
```

would process the file "file1" with the preprocessor before formatting it with *troff*.

SEE ALSO

cw(1), mm(1), mmt(1), nroff(1), tbl(1), troff(1), eqnchar(5), mm(5), mv(5).

Typesetting Mathematics—User's Guide by B. W. Kernighan and L. L. Cherry.

BUGS

To embolden digits, parentheses, etc., it is necessary to quote them, as in **bold "12.3"**.

See also *BUGS* under *troff*(1).

NAME

ex, edit − text editor

SYNOPSIS

ex [−] [−**v**] [−**t** tag] [−**r**] [+*command*] name ...
edit [ex options]

DESCRIPTION

Ex is the root of a family of editors: *edit*, *ex* and *vi*. *Ex* is a superset of *edit*, with the most notable extension being a display editing facility. Display based editing is the focus of *vi*.

If you have not used *ed*, or are a casual user, you will find that the editor *edit* is convenient for you. It avoids some of the complexities of *ex* used mostly by systems programmers and persons very familiar with *ed*.

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(1), which is a command which focuses on the display editing portion of *ex*.

The following options are recognized:

− suppresses all interactive-user feedback, as when processing editor scripts in command files.

−**v** Equivalent to using *vi* rather than *ex*.

−**t** Equivalent to an initial *tag* command, editing the file containing the *tag* and positioning the editor at its definition.

−**r** Used in recovering after an editor or system crash, retrieving the last saved version of the named file. If no file is specified, a list of saved files will be reported.

+*command*
Indicates that the editor should begin by executing the specified command. If *command* is omitted, then it defaults to $, positioning the editor at the last line of the first file initially. Other useful commands here are scanning patterns of the form /pat or line numbers, e.g., +100 to start at line 100.

Name arguments indicate files to be edited.

Documentation

The document, *Edit: A tutorial*, provides a comprehensive introduction to *edit* assuming no previous knowledge of computers or the UNIX system.

The *Ex Reference Manual* is a comprehensive and complete manual for the command mode features of *ex*, but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of *ex*, see the editing documents written by Brian Kernighan for the editor *ed*; the material in the introductory and advanced documents works also with *ex*.

An Introduction to Display Editing with Vi introduces the display editor *vi* and provides reference material on *vi*. In addition, the *Vi Quick Reference* card summarizes the commands of *vi* in a useful, functional way, and is useful with the *Introduction*.

FILES

/usr/lib/ex3.7strings	error messages
/usr/lib/ex3.7recover	recover command
/usr/lib/ex3.7preserve	preserve command
/etc/termcap	describes capabilities of terminals
~/.exrc	editor startup command file, user-created in home directory
/tmp/EX*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory
/usr/lib/tags	standard editor tag file

EXAMPLE

ex text

would invoke the editor with the file named "text".

SEE ALSO

awk(1), ed(1), grep(1), sed(1), vi(1)

BUGS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors don't print a name if the command line "−" option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files, and cannot appear in resultant files.

AUTHOR

William Joy and Mark Horton.

NAME

expr − evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

expr \| *expr*
returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr \& *expr*
returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { =, \>, \>=, \<, \<=, != } *expr*
returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, − } *expr*
addition or subtraction of integer-valued arguments.

expr { *, /, % } *expr*
multiplication, division, or remainder of the integer-valued arguments.

expr : *expr*
The matching operator : compares the first argument with the second argument which must be a regular expression; regular expression syntax is the same as that of *ed*(1), except that all patterns are "anchored" (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

EXAMPLE

```
a=`expr $a + 1`
```

adds 1 to the shell variable **a**.

```
# 'For $a equal to either "/usr/abc/file" or just "file"'
expr $a : '.*/\(.*\)' \| $a
```

returns the last segment of a path name (i.e., "file"). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

```
# A better representation of above example
expr //$a : '.*/\(.*\)'
```

the addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

```
expr $VAR : '.*'
```

returns the number of characters in **$VAR**.

SEE ALSO

ed(1), sh(1).

EXIT CODE

As a side effect of expression evaluation, *expr* returns the following exit values:

0 if the expression is neither null nor **0**
1 if the expression *is* null or **0**
2 for invalid expressions.

DIAGNOSTICS

syntax error for operator/operand errors
non-numeric argument if arithmetic is attempted on such a string

BUGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If **$a** is an =, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

```
expr X$a = X=
```

NAME

exterr - turn on/off the extended errors in the specified device

SYNOPSIS

exterr /dev/devicename [yn]

DESCRIPTION

Exterr turns on (or off) the reporting of extended errors on the specified device.

If reporting of errors is turned "off" with the argument **n**, only fatal errors are reported.

The default condition is "yes", in which case soft as well as hard errors are reported on the specified device. The devicename must be the "raw" one to access the *ioctl*.

EXAMPLE

```
exterr /dev/xxxx n
```

turns to off the reporting of extended errors for device **/dev/xxxx**.

NAME

factor — factor a number

SYNOPSIS

factor [number]

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{56} (about 7.2×10^{16}) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to $\sqrt{n}$ and occurs when n is prime or the square of a prime. It takes 30 seconds to factor a prime near 10^{14} on a 68000.

DIAGNOSTICS

"Ouch" for input out of range or for garbage input.

NAME

file − determine file type

SYNOPSIS

file [**−c**] [**−f** ffile] [**−m** mfile] arg ...

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable **a.out**, *file* will print the version stamp, provided it is greater than 0 (see *ld*(1)).

If the **−f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

File uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of **/etc/magic** explains its format.

The **−m** option instructs *file* to use an alternate magic file.

The **−c** flag causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **−c**.

EXAMPLE

file textfile programfile directory

reports the file names and directory name, and whether the files are English text, *nroff* input, a C program, or whatever.

NAME

find − find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each path name in the *path-name-list* (i.e., one or more path names) seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where +*n* means more than *n*, −*n* means less than *n* and *n* means exactly *n*.

−name *file* True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for **[**, **?** and ***)**.

−perm *onum* True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared:

(flags&onum)==onum

−type *c* True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **p**, or **f** for block special file, character special file, directory, fifo (a.k.a named pipe), or plain file.

−links *n* True if the file has *n* links.

−user *uname* True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.

−group *gname* True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.

−size *n* True if the file is *n* blocks long (512 bytes per block).

−atime *n* True if the file has been accessed in *n* days.

−mtime *n* True if the file has been modified in *n* days.

−ctime *n* True if the file has been changed in *n* days.

−exec *cmd* True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument **{}** is replaced by the current path name.

−ok *cmd* Like **−exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.

−print Always true; causes the current path name to be printed.

−cpio *device* Write the current file on *device* in *cpio*(4) format (5120 byte records).

−newer *file* True if the current file has been modified more recently than the argument *file*.

(*expression*) True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

1) The negation of a primary (! is the unary *not* operator).

2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

3) Alternation of primaries (**−o** is the *or* operator).

EXAMPLE

```
find / -perm 755 -exec ls "{}" ";"
```

will find all files, starting with the root directory, on which the permission levels have been set to 755 (see *chmod*(1)).

With **−exec** and a command such as *ls*, it is often necessary to escape the "{}" that stores the current pathname under investigation by putting it in double quotes. It is **always** necessary to escape the semicolon at the end of an **−exec** sequence.

Note again that it is also necessary to escape parentheses " \(" and " \) " used for grouping primaries, by means of a backslash.

```
find / \( −name a.out −o −name '*.o' \) −atime +7 −exec rm {} \;
```

removes all files named "a.out" or "*.o" that have not been accessed for a week.

FILES

/etc/passwd, /etc/group

SEE ALSO

cpio(1), sh(1), test(1), stat(2), cpio(4), fs(4).

NAME

freq – report on character frequencies in a file

SYNOPSIS

freq [file ...]

DESCRIPTION

Freq counts occurrences of characters in the list of files specified on the command line. If no files are specified, the standard input is read.

EXAMPLE

```
freq filea
```

will list a count of each character that appears in "filea".

NAME

fsplit − split fortran, ratfor, or efl files

SYNOPSIS

fsplit options files

DESCRIPTION

Fsplit splits the named *file(s)* into separate files, with one procedure per file. A procedure includes *blockdata*, *function*, *main*, *program*, and *subroutine* program segments. Procedure *X* is put in file *X* **.f**, *X* **.r**, or *X*.e depending on the language option chosen, with the following exceptions: *main* is put in the file *MAIN*.**[efr]** and unnamed *blockdata* segments in the files *blockdataN*.**[efr]** where *N* is a unique integer value for each file.

The following *options* pertain:

−f (default) Input files are *fortran*.

−r Input files are *ratfor*.

−e Input files are *Efl*.

−s Strip *fortran* input lines to 72 or fewer characters with trailing blanks removed.

SEE ALSO

csplit(1), efl(1), fortran(1), split(1).

NAME

get − get a version of an SCCS file

SYNOPSIS

get [−rSID] [−ccutoff] [−ilist] [−xlist] [−aseq-no.] [−**k**] [−**e**] [−**l**[p]] [−**p**] [−**m**] [−**n**] [−**s**] [−**b**] [−**g**] [−**t**] file ...

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with −. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also *FILES*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

−r*SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the −**e** keyletter is also used), as a function of the SID specified.

−c*cutoff* *Cutoff* date-time, in the form: YY[MM[DD[HH[MM[SS]]]]]
No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **−c7502** is equivalent to **−c750228235959**. Any number of non-numeric characters may separate the various 2 digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: "**−c77/2/2 9:22:25**". Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

```
~!get "−c%E% %U%" s.file
```

−**e** Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The −**e** keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *get*s for editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **get** −**e** for different SIDs is always allowed.

If the *g-file* generated by *get* with an −**e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the −**k** keyletter in place of

the **−e** keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the **−e** keyletter is used.

−b Used with the **−e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.

−i*list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID − SID

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

−x*list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **−i** keyletter for the *list* format.

−k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **−k** keyletter is implied by the **−e** keyletter.

−l[p] Causes a delta summary to be written into an *l-file*. If **−lp** is used, then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.

−p Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **−s** keyletter is used, in which case it disappears.

−s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

−m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

−n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **−m** and **−n** keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the **−m** keyletter generated format.

−g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

−t Used to access the most recently created ("top") delta in a given release (e.g., **−r1**), or release and level (e.g., **−r1.2**).

−a*seq-no.* The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(5)). This keyletter is used by the *comb*(1) command; it is not a generally useful keyletter, and users should not use it. If both the **−r** and **−a** keyletters are specified, the **−a** keyletter is used. Care should be taken when using the **−a** keyletter in conjunction with the **−e** keyletter, as the SID of the delta to be created may not be what one expects. The **−r** keyletter can be used with the **−a** and **−e** keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **−e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **−i** keyletter is used included deltas are listed following the notation "Included"; if the **−x** keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	**−b** Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	—	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	—	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	—	Trunk succ. in release ⩾ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	—	Branch succ.	R.L.B.S	R.L.(mB+1).1

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means

"maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

Successor.

† The **−b** keyletter is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of − means "irrelevant".

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	*Value*
%M%	Module name: either the value of the **m** flag in the file (see *admin*(1)), or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file (see *admin*(1)).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file (see *admin*(1)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as "this shouldn't have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string **@(#)** recognizable by *what*(1).
%W%	A shorthand notation for constructing *what*(1) strings for the UNIX System program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing *what*(1) strings for non-UNIX system program files. %A% = %Z%%Y% %M% %I%%Z%

EXAMPLE

get -e s.file1

generates from the SCCS format file, "s.file1", the text file, "file1", for editing.

FILES

Several auxiliary files may be created by *get*, These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **−p** keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **−k** keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **−l** keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
   * otherwise.
b. A blank character if the delta was applied or wasn't applied and ignored;
   * if the delta wasn't applied and wasn't ignored.
c. A code indicating a "special" reason why the delta was or was not applied:
   "I": Included.
   "X": Excluded.
   "C": Cut off (by a **−c** keyletter).
d. Blank.
e. SCCS identification (SID).
f. Tab character.
g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.
i. Login name of person who created *delta*.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an **−e** keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an **−e** keyletter for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it

is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **−i** keyletter argument if it was present, followed by a blank and the **−x** keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

SEE ALSO

admin(1), delta(1), help(1), prs(1), what(1), sccsfile(4).
"Source Code Control System"

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, then only one file may be named when the **−e** keyletter is used.

NAME

getopt − parse command options

SYNOPSIS

set −− `getopt** optstring **$*`

DESCRIPTION

Getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *Optstring* is a string of recognized option letters (see *getopt*(3C)); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option −− is used to delimit the end of the options. If it is used explicitly, *getopt* will recognize it; otherwise, *getopt* will generate it; in either case, *getopt* will place it at the end of the options. The shell's positional parameters ($1 $2 ...) are reset so that each option is preceded by a − and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set −− `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        −a | −b)        FLAG=$i; shift;;
        −o)             OARG=$2; shift 2;;
        −−)             shift; break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd −aoarg file file
cmd −a −o arg file file
cmd −oarg −a file file
cmd −a −oarg −− file file
```

SEE ALSO

sh(1), getopt(3C).

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NAME

greek – select terminal filter

SYNOPSIS

greek [–Tterminal]

DESCRIPTION

Greek is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character TELETYPE® Teletypewriter Model 37 terminal (which is the *nroff* default terminal) for certain other terminals. Special characters are simulated by overstriking, if necessary and possible. If the argument is omitted, *greek* attempts to use the environment variable **$TERM** (see *environ*(5)). The following *terminal*s are recognized currently:

300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	Diablo 1620 (alias DASI 450).
1620-12	Diablo 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	Tektronix 4014.
hp	Hewlett-Packard 2621, 2640, and 2645.
tek	Tektronix 4014.

EXAMPLE

```
nroff filename | greek -T4014
```

reinterprets the extended character set on a Tektronix 4014 terminal.

FILES

/usr/bin/300
/usr/bin/300s
/usr/bin/4014
/usr/bin/450
/usr/bin/hp

SEE ALSO

300(1), 4014(1), 450(1), eqn(1), hp(1), mm(1), nroff(1), tplot(1G), environ(5), greek(5), term(5).

NAME

grep, egrep, fgrep − search a file for a pattern

SYNOPSIS

grep [options] expression [files]

egrep [options] [expression] [files]

fgrep [options] [strings] [files]

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular *expressions* in the style of *ed*(1); it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

−v All lines but those matching are printed.

−x (Exact) only lines matched in their entirety are printed (*fgrep* only).

−c Only a count of matching lines is printed.

−l Only the names of files with matching lines are listed (once), separated by new-lines.

−n Each line is preceded by its relative line number in the file.

−b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

−s The error messages produced for nonexistent or unreadable files are suppressed (*grep* only).

−e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a − (does not work with *grep*).

−f *file*
The regular *expression* (*egrep*) or *strings* list (*fgrep*) is taken from the *file*.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters **$**, *, **[**, ^, **|**, **(**, **)**, and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'.

Fgrep searches for lines that contain one of the *strings* separated by new-lines.

Egrep accepts regular expressions as in *ed*(1), except for \(and \), with the addition of:

1. A regular expression followed by + matches one or more occurrences of the regular expression.
2. A regular expression followed by ? matches 0 or 1 occurrences of the regular expression.
3. Two regular expressions separated by | or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses () for grouping.

The order of precedence of operators is [], then * ? +, then concatenation, then | and new-line.

EXAMPLE

```
grep -v -c 'regular' grep.1
```

reports a count of the number of lines that do **not** contain the word *regular* in the file "grep.1".

SEE ALSO

ed(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Ideally there should be only one *grep*, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.
Lines are limited to 256 characters; longer lines are truncated.
Egrep does not recognize ranges, such as **[a−z]**, in character classes.

NAME

head − give first few lines

SYNOPSIS

head [−count] [file ...]

DESCRIPTION

This filter gives the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted it defaults to 10.

EXAMPLE

```
head -6 filea fileb filec
```

will print out the first six lines of the three specified files. The filename will appear before each new set of head lines listed, if more than one file has been specified.

SEE ALSO

tail(1).

NAME

help – ask for help in using SCCS

SYNOPSIS

help [args]

DESCRIPTION

Help finds information to explain a message from an SCCS command or explain the use of an SCCS command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., **ge6**, for message 6 from the *get* command).

type 2 Does not contain numerics (as a command, such as **get**)

type 3 Is all numeric (e.g., **212**)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

EXAMPLE

```
help he2
```

prints the message for error number "he2".

FILES

/usr/lib/help	directory containing files of message text.
/usr/lib/help/helploc	file containing locations of help files not in **/usr/lib/help**.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

hex − translates object files

SYNOPSIS

hex [−**f**] [−**l**] [−**n#**] [−**r**] [−**s0**] [−**s2**] [−**ns8**] [+**saddr**] ifile

DESCRIPTION

Hex translates object files into ASCII formats suitable for Motorola S-record downloading. The following options determine locations:

f The file specified is to be shipped as is without treating it as an object file.

l Output "Loading at" message.

n# Number of characters to output per record. # is a decimal number.

r Output a carriage return at the end of each S-record (instead of a newline).

s0 Output a leading **s0** record.

s2 **S2** records only (no **s1** records are produced).

ns8 Do not output a trailing **s8** (**s9**) record.

saddr Starting load address (in hex).

ifile File to be downloaded. The file's starting address is used if **saddr** is not present.

EXAMPLE

```
hex objfile
```

where "objfile" is the object file to be downloaded.

AUTHOR

Jeff Schriebman

NAME

hostname — set or print name of current host system

SYNOPSIS

hostname [nameofhost]

DESCRIPTION

The *hostname* command prints the name of the current host, as given before the "login" prompt. The super-user can set the hostname by giving an argument; this is usually done in the startup script **/etc/rc.**

SEE ALSO

gethostname(2N), sethostname(2N).

NAME

hp − handle special functions of HP 2640 and 2621-series terminals

SYNOPSIS

hp [**−e**] [**−m**]

DESCRIPTION

Hp supports special functions of the Hewlett-Packard 2640 series of terminals, with the primary purpose of producing accurate representations of most *nroff* output.

Regardless of the hardware options on your terminal, *hp* tries to do sensible things with underlining and reverse line-feeds. If the terminal has the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it has the "mathematical-symbol" feature, Greek and other special characters can be displayed.

The flags are as follows:

−e It is assumed that your terminal has the "display enhancements" feature, and so maximal use is made of the added display modes. Overstruck characters are presented in the Underline mode. Superscripts are shown in Half-bright mode, and subscripts in Half-bright, Underlined mode. If this flag is omitted, *hp* assumes that your terminal lacks the "display enhancements" feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in Inverse Video mode, i.e., dark-on-light, rather than the usual light-on-dark.

−m Requests minimization of output by removal of new-lines. Any contiguous sequence of 3 or more new-lines is converted into a sequence of only 2 new-lines; i.e., any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other special characters, *hp* provides the same set as does *300*(1), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown. The display is adequate for examining output from *neqn*.

DIAGNOSTICS

line too long if the representation of a line exceeds 1,024 characters.

The exit codes are **0** for normal termination, **2** for all errors.

EXAMPLE

```
nroff −h filea ... | hp
```

will *nroff* "filea" according to the special functions of the Hewlett-Packard 2640 series of terminals.

SEE ALSO

300(1), col(1), eqn(1), greek(1), nroff(1), tbl(1).

BUGS

An "overstriking sequence" is defined as a printing character followed by a backspace followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in Inverse Video; otherwise, only the first printing character is shown (again, underlined or in Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (e.g., reverse line-feeds, backspaces) can make text

"disappear"; in particular, tables generated by *tbl*(1) that contain vertical lines will often be missing the lines of text that contain the "foot" of a vertical line, unless the input to *hp* is piped through *col*(1).

Although some terminals do provide numerical superscript characters, no attempt is made to display them.

NAME

hpio − HP 2645A terminal tape file archiver

SYNOPSIS

hpio −**o**[**rc**] file ...

hpio −**i**[**rta**] [−**n** count]

DESCRIPTION

Hpio is designed to take advantage of the tape drives on Hewlett Packard 2645A terminals. Up to 255 UNIX System files can be archived onto a tape cartridge for off-line storage or for transfer to another UNIX System. The actual number of files depends on the sizes of the files. One file of about 115,000 bytes will almost fill a tape cartridge. Almost 300 1-byte files will fit on a tape, but the terminal will not be able to retrieve files after the first 255. This manual page is not intended to be a guide for using tapes on HP 2645A terminals, but tries to give enough information to be able to create and read tape archives and to position a tape for access to a desired file in an archive.

The **−o** (copy out) option copies the specified *file*(s), together with path name and status information to a tape drive on your terminal (which is assumed to be positioned at the beginning of a tape or immediately after a tape mark). The left tape drive is used by default. Each *file* is written to a separate tape file and terminated with a tape mark. When *hpio* finishes, the tape is positioned following the last tape mark written.

The **−i** (copy in) option extracts a file(s) from a tape drive (which is assumed to be positioned at the beginning of a file that was previously written by a **hpio −o**). The default action extracts the next file from the left tape drive.

Hpio always leaves the tape positioned after the last file read from or written to the tape. Tapes should always be rewound before the terminal is turned off. To rewind a tape depress the green function button, then function key 5, and then select the appropriate tape drive by depressing either function key 5 for the left tape drive or function key 6 for the right. If several files have been archived onto a tape, the tape may be positioned at the beginning of a specific file by depressing the green function button, then function key 8, followed by typing the desired file number (1−255) with no RETURN, and finally function key 5 for the left tape or function key 6 for the right. The desired file number may also be specified by a signed number relative to the current file number.

The meanings of the available options are:

r Use the right tape drive.

c Include a checksum at the end of each *file*. The checksum is always checked by **hpio −i** for each file written with this option by **hpio −o**.

n *count* The number of input files to be extracted is set to *count*. If this option is not given, *count* defaults to 1. An arbitrarily large *count* may be specified to extract all files from the tape. *Hpio* will stop at the end of data mark on the tape.

t Print a table of contents only. No files are created. Printed information gives the file size in bytes, the file name, the file access modes, and whether or not a checksum is included for the file.

a Ask before creating a file. **Hpio −i** normally prints the file size and name, creates and reads in the file, and prints a status message when the file has been read in. If a checksum is included with the file, it reports whether the checksum matched its computed value. With this option, the file size and name are printed followed by a **?**. Any response beginning with **y** or **Y** will cause the file to be copied in as above. Any other response will cause the file to be skipped.

FILES

/dev/tty?? to block messages while accessing a tape

SEE ALSO

2645A Display Station User's Manual , Hewlett-Packard Company, Part Number 02645-90001.

DIAGNOSTICS

BREAK
An interrupt signal terminated processing.

Can't create
file . File system access permissions did not allow *file* to be created.

Can't get tty options on stdout.
Hpio was unable to get the input-output control settings associated with the terminal.

Can't open
file . *File* could not be accessed to copy it to tape.

End of Tape.
No tape record was available when a read from a tape was requested. An end of data mark is the usual reason for this, but it may also occur if the wrong tape drive is being accessed and no tape is present.

"*file*" not a regular file.
File is a directory or other special file. Only regular files will be copied to tape.

Readcnt = *rc*, termcnt = *tc*.
Hpio expected to read *rc* bytes from the next block on the tape, but the block contained *tc* bytes. This is caused by having the tape improperly positioned or by a tape block being mangled by interference from other terminal I/O.

Skip to next file failed.
An attempt to skip over a tape mark failed.

Tape mark write failed.
An attempt to write a tape mark at the end of a file failed.

Write failed.
A tape write failed. This is most frequently caused by specifying the wrong tape drive, running off the end of the tape, or trying to write on a tape that is write protected.

WARNINGS

Tape I/O operations may copy bad data if any other I/O involving the terminal occurs. Do not attempt any type ahead while *hpio* is running. *Hpio* turns off write permissions for other users while it is running, but processes started asynchronously from your terminal can still interfere. The most

common indication of this problem, while a tape is being written, is the appearance of characters on the display screen that should have been copied to tape.

The keyboard, including the terminal's BREAK key, is locked during tape write operations; the BREAK key is only functional between writes.

Hpio must have complete control of the attributes of the terminal to communicate with the tape drives. Interaction with commands such as *cu*(1C) may interfere and prevent successful operation.

BUGS

Some binary files contain sequences that will confuse the terminal.

An **hpio −i** that encounters the end of data mark on the tape (e.g., scanning the entire tape with **hpio −itn 300**), leaves the tape positioned *after* the end of data mark. If a subsequent **hpio −o** is done at this point, the data will not be retrievable. The tape must be repositioned manually using the terminal's FIND FILE −1 operation (depress the green function button, function key 8, and then function key 5 for the left tape or function key 6 for the right tape) before the **hpio −o** is started.

If an interrupt is received by *hpio* while a tape is being written, the terminal may be left with the keyboard locked. If this happens, the terminal's RESET TERMINAL key will unlock the keyboard.

NAME

hyphen – find hyphenated words

SYNOPSIS

hyphen [files]

DESCRIPTION

Hyphen finds all the hyphenated words ending lines in *files* and prints them on the standard output. If no arguments are given, the standard input is used; thus, *hyphen* may be used as a filter.

EXAMPLE

If the file "text.hyphen" contains the following text:

```
This is an ex-
ample of the command hy-
phen, a com-
mand which finds all hyphen-
ated words in files and prints them on stan-
dard out-
put.
```

then

```
hyphen text.hyphen
```

will return

```
ex-ample
hy-phen
com-mand
hyphen-ated
stan-dard
out-put
```

SEE ALSO

mm(1), troff(1).

BUGS

Hyphen can't cope with hyphenated *italic* (i.e., underlined) words; it will often miss them completely, or mangle them.

Hyphen occasionally gets confused, but with no ill effects other than spurious extra output.

NAME

id – print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

Id writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

EXAMPLE

```
id guest
```

will return

```
uid=100 (guest) gid=100 (users)
```

where "100" and "guest" are the user's ID number and name and "100" and "users" are the user's group ID number and group name. These values are set up in the administrative file **/etc/passwd**.

SEE ALSO

logname(1), getuid(2).

NAME

ipcrm − remove a message queue, semaphore set or shared memory id

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

Ipcrm will remove one or more specified message, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

−q *msqid* removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.

−m *shmid* removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

−s *semid* removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.

−Q *msgkey* removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.

−M *shmkey* removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

−S *semkey* removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in *msgctl*(2), *shmctl*(2), and *semctl*(2). The identifiers and keys may be found by using *ipcs*(1).

SEE ALSO

ipcs(1), msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2).

NAME

ipcs − report inter-process communication facilities status

SYNOPSIS

ipcs [options]

DESCRIPTION

Ipcs prints certain information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following *options*:

−q Print information about active message queues.
−m Print information about active shared memory segments.
−s Print information about active semaphores.

If any of the options **−q**, **−m**, or **−s** are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed.

−b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.
−c Print creator's login name and group name. See below.
−o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)
−p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.
−t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, last *semop*(2) on semaphores.) See below.
−a Use all print *options*. (This is a shorthand notation for **−b**, **−c**, **−o**, **−p**, and **−t**.)
−C *corefile*
Use the file *corefile* in place of **/dev/kmem**.
−N *namelist*
The argument will be taken as the name of an alternate *namelist* (**/unix** is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

T (all) Type of the facility:
q message queue;
m shared memory segment;
s semaphore.

ID (all) The identifier for the facility entry.

KEY (all) The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)

MODE (all) The facility access modes and flags. The mode consists of 11 characters that are interpreted as follows:

The first two characters are:

- **R** if a process is waiting on a *msgrcv*;
- **S** if a process is waiting on a *msgsnd*;
- **D** if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;
- **C** if the associated shared memory segment is to be cleared when the first attach is executed;
- **–** if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

- **r** if read permission is granted;
- **w** if write permission is granted;
- **a** if alter permission is granted;
- **–** if the indicated permission is *not* granted.

OWNER (all) The login name of the owner of the facility entry.

GROUP (all) The group name of the group of the owner of the facility entry.

CREATOR (a,c) The login name of the creator of the facility entry.

CGROUP (a,c) The group name of the group of the creator of the facility entry.

CBYTES (a,o) The number of bytes in messages currently outstanding on the associated message queue.

QNUM (a,o) The number of messages currently outstanding on the associated message queue.

QBYTES (a,b) The maximum number of bytes allowed in messages outstanding on the associated message queue.

LSPID (a,p) The process ID of the last process to send a message to the associated queue.

LRPID (a,p) The process ID of the last process to receive a message from the associated queue.

STIME (a,t) The time the last message was sent to the associated queue.

RTIME	(a,t)	The time the last message was received from the associated queue.
CTIME	(a,t)	The time when the associated entry was created or changed.
NATTCH	(a,o)	The number of processes attached to the associated shared memory segment.
SEGSZ	(a,b)	The size of the associated shared memory segment.
CPID	(a,p)	The process ID of the creator of the shared memory entry.
LPID	(a,p)	The process ID of the last process to attach or detach the shared memory segment.
ATIME	(a,t)	The time the last attach was completed to the associated shared memory segment.
DTIME	(a,t)	The time the last detach was completed on the associated shared memory segment.
NSEMS	(a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME	(a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/unix	system namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

msgop(2), semop(2), shmop(2).

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

NAME

join − relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is −, the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

Fields are normally separated by blank, tab or new-line. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

−a*n* In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.

−e *s* Replace empty output fields by string *s*.

−j*n m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file.

−o *list* Each output line comprises the fields specified in *list*, each element of which has the form *n*. *m*, where *n* is a file number and *m* is a field number.

−t *c* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant.

EXAMPLE

If "file1" contains:

```
Austen -
Bailey -
Clark -
Dawson -
Smith -
```

and "file2" contains:

```
Austen Jack Anchor Brewery
Clark Maryann Shoeshop
Daniels Steve Computer Software
Dawson Sylvia Toot Sweets
Smith Sally Talcum Powdery
```

then

```
join −j1 1 −j2 1 −o 2.2 2.1 1.2 2.3 2.4 file1 file2
```

will generate

```
Jack Austen - Anchor Brewery
Maryann Clark - Shoeshop
Sylvia Dawson - Toot Sweets
Sally Smith - Talcum Powdery
```

SEE ALSO

awk(1), comm(1), sort(1).

BUGS

With default field separation, the collating sequence is that of **sort −b**; with **−t**, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq* and *awk* (1) are wildly incongruous.

NAME

kill − terminate a process

SYNOPSIS

kill [−signo] PID ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with & is reported by the Shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(1).

The details of the kill are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the superuser.

If a signal number preceded by − is given as first argument, that signal is sent instead of terminate (see *signal*(2)). In particular "kill −9 ..." is a sure kill.

EXAMPLE

```
kill 24068
```

Sends signal 15 to the process with the ID number 24068.

SEE ALSO

ps(1), sh(1), kill(2), signal(2).

NAME

ld − link editor

SYNOPSIS

ld [option] file ...

DESCRIPTION

Ld combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object files are given, and *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the **−r** option must be given to preserve the relocation bits.) The output of *ld* is left on **a.out**. This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important.

The symbols "_etext", "_edata" and "_end" ("etext", "edata" and "end" in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several options. Except for **−l**, they should appear before the file names.

−s "Strip" the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by *strip*(1).

−u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

−l*x* This option is an abbreviation for the library name "**/lib/lib***x***.a**", where *x* is a string. If that does not exist, *ld* tries "**/usr/lib/lib***x***.a**". A library is searched when its name is encountered, so the placement of a **−l** is significant.

−x Do not preserve local (non-.globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

−X Save local symbols except for those whose names begin with "L". This option is used by *cc*(1) to discard internally generated labels while retaining symbols local to routines.

−r Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the "undefined symbol" diagnostics.

−R *x* Set starting relocation address of program to *x* (*x* is in hex).

−LT *x* Set the text relocation address to *x* (*x* is in hex).

−LD *x* Set the data relocation address to *x* (*x* is in hex).

−LC *x* Set the common relocation address to *x* (*x* is in hex).

−LB *x* Set the bss relocation address to *x* (*x* is in hex).

−d Force definition of common storage even if the **−r** flag is present.

−n Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible protection boundary following the end of the text.

−N *x* Set the data relocation boundary to *x* for shared text programs. The value *x* may be followed by a *k* or *K* to indicate multiplication by 1024.

−o The *name* argument after **−o** is used as the name of the *ld* output file, instead of **a.out**.

−e The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default.

−F *x* Add offset *x* to all data references (*x* is in hex).

EXAMPLE

```
ld -s /lib/crt0.o filea.o fileb.o -lc
```

will load subroutines "filea" with "fileb" for execution and remove its symbol table.

FILES

/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
a.out	default output file
/lib/crt0.o	"C" start up routine

SEE ALSO

ar(1), as(1), cc(1).

NAME

lex − generate programs for simple lexical tasks

SYNOPSIS

lex [**−rctvn**] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text.

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A file **lex.yy.c** is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in *yytext*, an external character array. Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in **[abx−z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators *, +, and ? mean respectively any non-negative number of, any positive number of, and either zero or one occurrences of, the previous character or character class. The character . is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported. The notation $r\{d,e\}$ in a rule indicates between d and e instances of regular expression r. It has higher precedence than |, but lower than *, ?, +, and concatenation. The character ^ at the beginning of an expression permits a successful match only immediately after a new-line, and the character $ at the end of an expression requires a trailing new-line. The character / in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within " symbols or preceded by \ . Thus **[a−zA−Z]+** matches a string of letters.

Three subroutines defined as macros are expected: **input()** to read a character; **unput(***c***)** to replace a character read; and **output(***c***)** to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named **yylex()**, and the library contains a **main()** which calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function **yymore()** accumulates additional characters into the same *yytext*; and the function **yyless(***p***)** pushes back the portion of the string matched beginning at p, which should be between *yytext* and *yytext*+*yyleng*. The macros *input* and *output* use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes %% , it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a %% , as in YACC. Lines preceding %% which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with {} . Note that curly brackets do not imply parentheses; only string substitution is done.

The external names generated by *lex* all begin with the prefix **yy** or **YY**.

The flags must appear before any files. The flag **−r** indicates RATFOR actions, **−c** indicates C actions and is the default, **−t** causes the **lex.yy.c** program to be written instead to standard output, **−v** provides a one-line

summary of statistics of the machine generated, **−n** will not print out the − summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%**p** *n*	number of positions is *n* (default 2000)
%**n** *n*	number of states is *n* (500)
%**t** *n*	number of parse tree nodes is *n* (1000)
%**a** *n*	number of transitions is *n* (3000)

The use of one or more of the above automatically implies the **−v** option, unless the **−n** option is used.

EXAMPLE

```
D       [0−9]
%%
if      printf("IF statement\n");
[a−z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input() != '*');
                switch (input())
                        {
                        case '/': break;
                        case '*': unput('*');
                        default: go to loop;
                        }
                }
```

SEE ALSO

yacc(1).

LEX−Lexical Analyzer Generator by M. E. Lesk and E. Schmidt.

BUGS

The **−r** option is not yet fully operational.

NAME

line − read one line

SYNOPSIS

line

DESCRIPTION

Line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on **EOF** and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

EXAMPLE

```
line
Hello world
```

will return

```
Hello world
```

In the Bourne shell (sh):

```
a=`line`
hi there
echo $a
```

will return

```
hi there
```

In the C-shell (csh):

```
set a=`line`
bye bye
echo $a
```

will return

```
bye bye
```

SEE ALSO

sh(1), read(2).

NAME

lint − a C program checker

SYNOPSIS

lint [**−abhlnpuvx**] file ...

DESCRIPTION

Lint attempts to detect features of the C program *files* which are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things which are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

It is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. By default, *lint* uses function definitions from the standard lint library **llib-lc.ln**; function definitions from the portable lint library **llib-port.ln** are used when *lint* is invoked with the **−p** option.

Any number of *lint* options may be used, in any order. The following options are used to suppress certain kinds of complaints:

−a Suppress complaints about assignments of long values to variables that are not long.

−b Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in a large number of such complaints.)

−h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

−u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program.)

−v Suppress complaints about unused arguments in functions.

−x Do not report variables referred to by external declarations but never used.

The following arguments alter *lint*'s behavior:

−l*x* Include additional lint library **llib-l***x***.ln**. You can include a lint version of the math library **llib-lm.ln** by inserting **−lm** on the command line. This argument does not suppress the default use of **llib-lc.ln**. This option can be used to keep local lint libraries and is useful in the development of multi-file projects. To generate llib-lX.ln from llib-lX, use:

```
% cc −E −C −Dlint llib-lX | /usr/lib/lint/lintl −vx −H/tmp/lint$$ > llib-lX.ln
% rm − f /tmp/lint$$
```

−n Do not check compatibility against either the standard or the portable lint library.

−p Attempt to check portability to other dialects (IBM and GCOS) of C.

The **−D**, **−U**, and **−I** options of *cc*(1) are also recognized as separate arguments.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
at appropriate points stops comments about unreachable code.

/*VARARGS*n*/
suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/
turns on the **−v** option for the next function.

/*LINTLIBRARY*/
at the beginning of a file shuts off complaints about unused functions in this file.

Lint produces its first output on a per source file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

EXAMPLE

```
lint −b myfile.c
```

checks the consistency of the file "myfile.c". The **−b** option indicates that unreachable **break** statements are not to be checked. This option might well be used on files that *lex*(1) generates.

FILES

/usr/lib/lint[12]	programs
/usr/lib/llib-lc.ln	declarations for standard functions (binary format; source is in **/usr/lib/llib-lc**)
/usr/lib/llib-port.ln	declarations for portable functions (binary format; source is in **/usr/lib/llib-port**)
/usr/lib/llib-lm.ln	declarations for standard math functions (binary format; source is in **/usr/lib/llib-lm**)
/usr/tmp/*lint*	temporaries

SEE ALSO

cc(1).

BUGS

Exit(2) and other functions which do not return are not understood; this causes various lies.

NAME

login − sign on

SYNOPSIS

login [name [env-var ...]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-d* to indicate an "end-of-file".

If *login* is invoked as a command, it must replace the initial command interpreter. This is accomplished by typing:

exec login

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful *login*.

If you do not complete the *login* successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful *login*, accounting files are updated, the procedure **/etc/profile** is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh* (1)) is initialized, and the file **.profile** in the working directory is executed, if it exists. These specifications are found in the **/etc/passwd** file entry for the user. The name of the command interpreter is − followed by the last component of the interpreter's pathname (i.e., **−sh**). If this field in the password file is empty, then the default command interpreter, **/bin/sh** is used.

The basic *environment* (see *environ* (5)) is initialized to:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
SHELL=*last-field-of-passwd-entry*
MAIL=/usr/mail/*your-login-name*
TZ=*timezone-specification*

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your *login* name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as

L *n*=xxx

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people,

logging into restricted shell environments, from spawning secondary shells which aren't restricted. Both *login* and *getty* understand simple single character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

EXAMPLE

At the beginning of each terminal session, the following sort of message is displayed on the screen:

```
UniSoft 68000 UNIX
:login:
```

to which a user name is the appropriate response.

FILES

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/*your-name*	mailbox for user *your-name*
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	systemwide personal profile (*sh*(1))
/etc/cshrc	systemwide personal csh startup (*csh*(1))
.profile	personal profile (*sh*(1))
.login	personal csh startup used at login time (*csh*(1))
.cshrc	personal csh startup (*csh*(1))
.logout	personal csh logout used at logout time (*csh*(1))

SEE ALSO

mail(1), newgrp(1), sh(1), su(1), passwd(4), profile(4), environ(5).

DIAGNOSTICS

Login incorrect
if the user name or the password cannot be matched.

No shell, cannot open password file, or *no directory*
consult a UNIX system programming counselor.

No utmp entry. You must exec "login" from the lowest level "sh".
if you attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

NAME

logname – get login name

SYNOPSIS

logname

DESCRIPTION

Logname returns the contents of the environment variable **$LOGNAME**, which is set when a user logs into the system.

EXAMPLE

```
logname
```

displays the **$LOGNAME** of the user logged into the system on the current port.

FILES

/etc/profile

SEE ALSO

env(1), login(1), logname(3X), environ(4), printenv(1).

NAME

lorder − find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive *files* (see *ar*(1)). The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1). Note that the link editor *ld*(1) is capable of multiple passes over an archive in the portable archive format (see *ar*(4)) and does not require that *lorder*(1) be used when building an archive. The usage of the *lorder*(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

EXAMPLE

```
ar cr library lorder *.o | tsort
```

builds a new library from existing **.o** files.

FILES

*symref, *symdef temporary files

SEE ALSO

ar(1), ld(1), tsort(1), ar(4).

BUGS

Object files whose names do not end with **.o**, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

NAME

lp, cancel − send/cancel requests to an LP line printer

SYNOPSIS

lp [−**c**] [−**d**dest] [−**m**] [−**n**number] [−**o**option] [−**s**] [−**t**title] [−**w**] files

cancel [ids] [printers]

DESCRIPTION

Lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name − stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

Lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see below) or find the status (see *lpstat*(1)) of the request.

The following options to *lp* may appear in any order and may be intermixed with file names:

−**c** Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the −**c** option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that in the absence of the −**c** option, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.

−**d** *dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted (see *accept*(1M) and *lpstat*(1)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(1)).

−**m** Send mail (see *mail*(1)) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

−**n** *number* Print *number* copies (default of 1) of the output.

−**o** *option* Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the −**o** keyletter more than once. For more information about what is valid for *options*, see *Models* in *lpadmin*(1M).

−**s** Suppress messages from *lp*(1) such as "request id is ...".

−**t** *title* Print *title* on the banner page of the output.

-w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

Cancel cancels line printer requests that were made by the *lp*(1) command. The command line arguments may be either request *ids* (as returned by *lp*(1)) or *printer* names (for a complete list, use *lpstat*(1)). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1).
accept(1M), lpadmin(1M), lpsched(1M) in the *UniPlus+ Administrator's Manual.*

NAME

lpr − line printer spooler

SYNOPSIS

lpr [option ...] [name ...]

DESCRIPTION

Lpr causes the named files to be queued for printing on a line printer. If no names appear, the standard input is assumed; thus *lpr* may be used as a filter.

The following *options* may be given (each as a separate argument and in any order) before any file name arguments:

−c Makes a copy of the file to be sent before returning to the user.
−r Removes the file after sending it.
−m When printing is complete, reports that fact by *mail*(1).
−n Does not report the completion of printing by *mail*(1). This is the default option.
−f*file* Use *file* as a dummy file name to report back in the mail. (This is useful for distinguishing multiple runs, especially when *lpr* is being used as a filter).

Please note that the directory **/usr/spool/lpd** must be owned by daemon and have mode 0755; **/bin/lpr** must have mode 4755; and **/dev/lp** must be owned by daemon and have mode 600.

EXAMPLE

```
cat asa | lpr
```

will print the file "asa" on the line printer.

FILES

/etc/passwd	user's identification and accounting data.
/usr/lib/lpd	line printer daemon.
/usr/spool/lpd/*	spool area.

SEE ALSO

lp(1).

NAME

lpstat − print LP status information

SYNOPSIS

lpstat [options]

DESCRIPTION

Lpstat prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *Lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

−u"user1, user2, user3"

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

lpstat −o

prints the status of all output requests.

−a[*list*] Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

−c[*list*] Print class names and their members. *List* is a list of class names.

−d Print the system default destination for *lp*.

−o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.

−p[*list*] Print the status of printers. *List* is a list of printer names.

−r Print the status of the LP request scheduler.

−s Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

−t Print all status information.

−u[*list*] Print status of output requests for users. *List* is a list of login names.

−v[*list*] Print the names of printers and the pathnames of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1).

NAME

ls − list contents of directories

SYNOPSIS

ls [**−logtasdrucifp**] names

DESCRIPTION

For each directory named, *ls* lists the contents of that directory; for each file named, *ls* repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents. There are several options:

−l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will contain the major and minor device numbers, rather than a size.

−o The same as **−l**, except that the group is not printed.

−g The same as **−l**, except that the owner is not printed.

−t Sort by time of last modification (latest first) instead of by name.

−a List all entries; usually entries beginning with '.' are suppressed, except for the super user. '.' and '..' are always suppressed if the **−a** option is not used.

−s Give size in blocks (including indirect blocks) for each entry.

−d If argument is a directory, list only its name; often used with **−l** to get the status of a directory.

−r Reverse the order of sort to get reverse alphabetic or oldest first, as appropriate.

−u Use time of last access instead of last modification for sorting (with the **−t** option) and/or printing (with the **−l** option).

−c Use time of last modification of the inode (mode, etc.) instead of last modification of the file for sorting (**−t**) and/or printing (**−l**).

−i For each file, print the i-number in the first column of the report.

−f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **−l**, **−t**, **−s**, and **−r**, and turns on **−a**; the order is the order in which entries appear in the directory.

−p Put a slash after each filename if that file is a directory. Especially useful for CRT terminals when combined with the *pr*(1) command as follows: **ls −p | pr −5 −t −w80**.

The mode printed under the **−l** option consists of 11 characters that are interpreted as follows:

The first character is:

d if the entry is a directory;
b if the entry is a block special file;
c if the entry is a character special file;
p if the entry is a fifo (a.k.a. "named pipe") special file;
− if the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
- if the indicated permission is *not* granted.

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise, the user-execute permission character is given as **s** if the file has set-user-ID mode. The last character of the mode (normally **x** or **-**) is **t** if the 1000 (octal) bit of the mode is on; see *chmod*(1) for the meaning of this mode. The indications of set-ID and 1000 bit of the mode are capitalized (**S** and **T** respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

EXAMPLE

ls **-l** /etc

will list all entries in **/etc** in long format.

FILES

/etc/passwd to get user IDs for **ls -l** and **ls -o**.
/etc/group to get group IDs for **ls -l** and **ls -g**.

SEE ALSO

chmod(1), find(1), ls7(1).

NAME

ls7 − list contents of directory (Berkeley version)

SYNOPSIS

ls7 [**−1ACFRabcdfgilmnqrstux**] name ...

DESCRIPTION

For each directory argument, *ls7* lists the contents of the directory; for each file argument, *ls7* repeats the file name(s) and any other information requested with the *ls7* options. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The format chosen depends on whether the output is going to a teletype, and may also be controlled by option flags. The default format for a teletype is to list the contents of directories in multi-column format, with the entries sorted down the columns. (Files which are not the contents of a directory being interpreted are always sorted across the page rather than down the page in columns. This is because the individual file names may be arbitrarily long.) Files are listed first, and each directory being listed is labeled with its pathname, when two or more directory listings are requested. If the standard output is not a teletype, the default format is to list one entry per line. Finally, there is a stream output format in which files are listed across the page, separated by "," characters. The **−m** flag enables this format.

There are numerous options:

−1 Force one entry per line output format, e.g., to a teletype.

−A reverses the default state of the **−a** option.

−C Force multi-column output, e.g., to a file or a pipe.

−F Cause directories to be marked with a trailing "/" and executable files to be marked with a trailing "*"; this is the default if the last character of the name the program is invoked with is a "f" (for example, by linking **/bin/ls7** to **/bin/lf**).

−R Recursively list subdirectories encountered.

−a List all entries; usually entries beginning with '.' are suppressed, except for the super user. '.' and '..' are always suppressed if the **−a** option is not used.

−b Force printing of non-graphic characters to be in the "\ddd" notation, in octal.

−c Use time of file creation for sorting (**−t**) or printing (**−l**).

−d If argument is a directory, list only its name, not its contents (mostly used with **−l** to get status on directory).

−f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **−l**, **−t**, **−s**, and **−r**, and turns on **−a**; the order is the order in which entries appear in the directory.

−g Give group ID instead of owner ID in long listing.

−i Print i-number in first column of the report for each file listed.

−l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file, the size field will instead contain the major and minor device numbers.

−m Force stream output format.

−n User and group numbers, rather than names, will be printed in long (**−l**) listings.

−q Force printing of non-graphic characters in file names as the character "?"; this normally happens only if the output device is a teletype.

−r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

−s Give size in blocks, including indirect blocks, for each entry.

−t Sort by time modified (latest first) instead of by name, as is normal.

−u Use time of last access instead of last modification for sorting (**−t**) or printing (**−l**).

−x Force columnar printing to be sorted across rather than down the page; this is the default if the last character of the name the program is invoked with is an "x" (for example, by linking **/bin/ls7** to **/bin/lx**).

The mode printed under the **−l** (long) option contains 11 characters which are interpreted as follows: (see also *chmod*(1)). The first character is:

d if the entry is a directory;
b if the entry is a block-type special file;
c if the entry is a character-type special file;
m if the entry is a multiplexor-type character special file;
− if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
− if the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise the user-execute permission character is given as **s** if the file has set-user-ID mode.

The last character of the mode (normally "x" or "−") is **t** if the 1000 bit of the mode is on. See *chmod*(1) for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

EXAMPLE

ls7

lists the contents of the current directory in multi-column format.

FILES

/etc/passwd to get user and group IDs given in **ls7 −l**.

BUGS

Newline and tab are considered printing characters in file names.
The output device is assumed to be 80 columns wide.
Column widths choices are poor for terminals which can tab.

NAME

m4 − macro processor

SYNOPSIS

m4 [options] [files]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is −, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

−e Operate interactively. Interrupts are ignored and the output is unbuffered. Using this mode requires a special state of mind.

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

−B *int* Change the size of the push-back and argument collection buffers from the default of 4,096.

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

−S *int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

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

To be effective, these flags must appear before any file names and before any **−D** or **−U** flags:

−D *name*[= *val*]
Defines *name* to *val* or to null in *val*'s absence.

−U *name*
undefines *name*.

Macro calls have the form:

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

The (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 new-lines 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 which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

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

undefine — removes the definition of the macro named in its argument.

defn — returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

pushdef — like *define*, but saves any previous definition.

popdef — removes current definition of its argument(s), exposing the previous one if any.

ifdef — if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on the UNIX System versions of *m4*.

shift — returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

changequote — change quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., ``').

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

divert — *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. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

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

divnum — returns the value of the current output stream.

dnl — reads and discards characters up to and including the next new-line.

ifelse — has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise,

	the value is either the fourth string, or, if it is not present, null.
incr	returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
decr	returns the value of its argument decremented by 1.
eval	evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~ ; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
len	returns the number of characters in its argument.
index	returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.
substr	returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
translit	transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	returns the contents of the file named in the argument.
sinclude	is identical to *include*, except that it says nothing if the file is inaccessible.
syscmd	executes the UNIX System command given in the first argument. No value is returned.
sysval	is the return code from the last call to *syscmd*.
maketemp	fills in a string of XXXXX in its argument with the current process ID.
m4exit	causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.
m4wrap	argument 1 will be pushed back at final EOF; example: m4wrap('cleanup()')
errprint	prints its argument on the diagnostic output file.
dumpdef	prints current names and definitions, for the named items, or for all if no arguments are given.
traceon	with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
traceoff	turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

EXAMPLE

```
m4 file1 file2 > outputfile
```

will run the *m4* macro processor on the files "file1" and "file2", redirecting the output into "outputfile".

SEE ALSO

cc(1), cpp(1).
The M4 Macro Processor by B. W. Kernighan and D. M. Ritchie.

NAME

m68k, pdp11, u3b, vax – provide truth value about your processor type

SYNOPSIS

m68k

pdp11

u3b

vax

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

m68k True if you are on a 68000.

pdp11 True if you are on a PDP-11/45 or PDP-11/70.

u3b True if you are on a 3B20S.

vax True if you are on a VAX-11/750 or VAX-11/780.

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

SEE ALSO

sh(1), test(1), true(1).

NAME

mail, rmail − send mail to users or read mail

SYNOPSIS

mail [**−epqr**] [**−f** file]

mail [**−t**] persons

rmail [**−t**] persons

DESCRIPTION

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

<new-line>	Go on to next message.
+	Same as <new-line>.
d	Delete message and go on to next message.
p	Print message again.
−	Go back to previous message.
s [*files*]	Save message in the named *files* (**mbox** is default).
w [*files*]	Save message, without its header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons* (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 do *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 *persons* are named, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a .) and adds it to each *person*'s *mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e., "From ...") are preceded with a >. The **−t** option causes the message to be preceded by all *persons* the *mail* is sent to. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the file **dead.letter** will be saved to allow editing and resending.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1C)). Everything after the first exclamation mark in *persons* is interpreted by the remote system. In particular, if *persons* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying **a!b!cde** as a recipient's name causes the message to be sent to user **b!cde** on system **a**. System **a** will

interpret that destination as a request to send the message to user **cde** on system **b**. This might be useful, for instance, if the sending system can access system **a** but not system **b**, and system **a** has access to system **b**.

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

Forward to *person*

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

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

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

EXAMPLE

mail carolyn

accepts whatever message is typed up to an EOF. Carolyn will be notified that she has mail the next time she logs in.

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

mail

FILES

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

SEE ALSO

login(1), uucp(1C), write(1).

BUGS

Race 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

make − maintain, update, and regenerate groups of programs

SYNOPSIS

make [−**f** makefile] [−**p**] [−**i**] [−**k**] [−**s**] [−**r**] [−**n**] [−**b**] [−**e**] [−**m**] [−**t**] [−**d**] [−**q**] [names]

DESCRIPTION

The following is a brief description of all options and some special names:

−**f** *makefile* Description file name. *Makefile* is assumed to be the name of a description file. A file name of − denotes the standard input. The contents of *makefile* override the built-in rules if they are present.

−**p** Print out the complete set of macro definitions and target descriptions.

−**i** Ignore error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.

−**k** When a command returns nonzero status, abandon work on the current entry, but continue on other branches that do not depend on that entry.

−**s** Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.

−**r** Do not use the built-in rules.

−**n** No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.

−**b** Compatibility mode for old makefiles.

−**e** Environment variables override assignments within makefiles.

−**m** Print a memory map showing text, data, and stack. This option is a no-operation on systems without the *getu* system call.

−**t** Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

−**d** Debug mode. Print out detailed information on files and times examined.

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

.DEFAULT If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.

.PRECIOUS Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT Same effect as the −**s** option.

.IGNORE Same effect as the −**i** option.

Make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **−f** option is present, **makefile**, **Makefile**, **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is **−**, the standard input is taken. More than one **−** *makefile* argument pair may appear.

Make updates a target only if it depends on files that are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out of date.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a :, then a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that begin with a tab are shell commands to be executed to update the target. The first line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash><new-line> sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (**#**) and new-line surround comments.

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o −o pgm
a.o: incl.h a.c
        cc −c a.c
b.o: incl.h b.c
        cc −c b.c
```

Command lines are executed one at a time, each by its own shell. The first one or two characters in a command can be the following: **−**, **@**, **−@**, or **@−**. If **@** is present, printing of the command is suppressed. If **−** is present, *make* ignores an error. A line is printed when it is executed unless the **−s** option is present, or the entry **.SILENT:** is in *makefile*, or unless the initial character sequence contains a **@**. The **−n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see discussion of the **MAKEFLAGS** macro under *Environment*). The **−t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the **−i** option is present, or the entry **.IGNORE:** appears in *makefile*, or the initial character sequence of the command contains **−**, the error is ignored. If the **−k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **-b** option allows old makefiles (those written for the old version of *make*) to run without errors. The difference between the old version of *make* and this version is that this version requires all dependency lines to have a (possibly null or implicit) command associated with them. The previous version of *make* assumed if no command was specified explicitly that the command was null.

Interrupt and quit cause the target to be deleted unless the target is a dependency of the special name **.PRECIOUS**.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **-e** option causes the environment to override the macro assignments in a makefile.

The **MAKEFLAGS** environment variable is processed by *make* as containing any legal input option (except **-f**, **-p**, and **-d**) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the **-n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make -n** recursively on a whole software system to see what would have been executed. This is because the **-n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Macros

Entries of the form *string1* = *string2* are macro definitions. *String2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of $(*string1*[:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

$*** The macro **$* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

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

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

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

or:

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

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

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

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **$?**. The reasons for this are debatable.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

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

The internal rules for *make* are contained in the source file **rules.c** for the *make* program. These rules can be locally modified. To print out the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used:

```
make -fp - 2>/dev/null </dev/null
```

The only peculiarity in this output is the **(null)** string which *printf*(3S) prints when handed a null string.

A tilde in the above rules refers to an SCCS file (see *sccsfile*(4)). Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix it is incompatible with *make*'s suffix point-of-view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e. **.c**:) is the definition of how to build *x* from *x*.**c**. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

```
.SUFFIXES: .o .c .y .l .s
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly:

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

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1) respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

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

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

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

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

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files

are out of date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

EXAMPLE

make CFLAGS=−O −f make.special

invokes *make* with command file "make.special" and redefines compiler options flag CFLAGS to be "−O".

FILES

[Mm]akefile and s.[Mm]akefile

SEE ALSO

sh(1).
Make−A Program for Maintaining Computer Programs by S. I. Feldman.
An Augmented Version of Make by E. G. Bradford.

BUGS

Some commands return non-zero status inappropriately; use **−i** to overcome the difficulty. Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across new-lines in *make*. The syntax **(lib(file1.o file2.o file3.o)** is illegal. You cannot build **lib(file.o)** from **file.o**. The macro **$(a:.o=.c~)** doesn't work.

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 lower-case 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)). Usually, its input and output will be pipes.

EXAMPLE

```
/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*.

SEE ALSO

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

NAME

man, manprog − print entries in this manual

SYNOPSIS

man [options] [section] titles

/usr/lib/manprog file

DESCRIPTION

Man locates and prints the entry of this manual named *title* in the specified *section*. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) The *title* is entered in lower case. The *section* number may not have a letter suffix. If no *section* is specified, the whole manual is searched for *title* and all occurrences of it are printed. *Options* and their meanings are:

- **−t** Typeset the entry in the default format (8.5"×11").
- **−s** Typeset the entry in the small format (6"×9").
- **−Tst** Directs the output to the MHCC STARE facility.
- **−T***term* Format the entry using *nroff* and print it on the standard output (usually, the terminal); *term* is the terminal type (see *term*(5) and the explanation below); for a list of recognized values of *term*, type **help term2**. The default value of *term* is **450**.
- **−w** Print on the standard output only the *path names* of the entries, relative to **/usr/man**, or to the current directory for **−d** option.
- **−d** Search the current directory rather than **/usr/man**; requires the full file name (e.g., **cu.1c**, rather than just **cu**).
- **−12** Indicates that the manual entry is to be produced in 12-pitch. May be used when **$TERM** (see below) 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 *man* to invoke *col*(1); note that *col*(1) is invoked automatically by *man* unless *term* is one of **300**, **300s**, **450**, **37**, **4000a**, **382**, **4014**, **tek**, **1620**, and **X**.
- **−y** Causes *man* to use the non-compacted version of the macros.

The above *options* other than **−d**, **−c**, and **−y** are mutually exclusive. Any other *options* are passed to *troff*, *nroff*, or the *man*(5) macro package.

When using *nroff*, *man* examines the environment variable **$TERM** (see *environ*(5)) and attempts to select options to *nroff*, as well as filters, that adapt the output to the terminal being used. The **−T***term* option overrides the value of **$TERM**; in particular, one should use **−Tlp** when sending the output of *man* to a line printer.

Section may be changed before each *title*.

If the first line of the input for an entry consists solely of the string:

'\"*x*

where *x* is any combination of the three characters **c**, **e**, and **t**, and where there is exactly one blank between the double quote ( ) and *x*, then *man* will preprocess its input through the appropriate combination of *cw*(1), *eqn*(1) (*neqn* for *nroff*) and *tbl*(1), respectively; if *eqn* or *neqn* are invoked, they will automatically read the file **/usr/pub/eqnchar** (see *eqnchar*(5)).

The *man* command executes *manprog* that takes a file name as its argument. *Manprog* calculates and returns a string of three register definitions used by the formatters identifying the date the file was last modified. The

returned string has the form:

−rd *day* **−rm** *month* **−ry** *year*

and is passed to *nroff* which sets this string as variables for the *man* macro package. Months are given from 0 to 11, therefore month is always 1 less than the actual month. The *man* macros calculate the correct month. If the *man* macro package is invoked as an option to *nroff*/*troff* (i.e., *nroff* **−***man file*), then the current day/month/year is used as the printed date.

EXAMPLE

man man

would reproduce on the terminal this entry, as well as any other entries named "man" that may exist in other sections of the manual, e.g., *man*(5).

FILES

/usr/man/u_man/man[1-6]/*	the *UniPlus*+ *User's Manual*
/usr/man/a_man/man[178]/*	the *UniPlus*+ *Administrator's Manual*
/usr/man/local/man[1-8]/*	local additions
/usr/lib/manprog	calculates modification dates of entries

SEE ALSO

cw(1), eqn(1), nroff(1), tbl(1), troff(1), environ(5), man(5), term(5).

BUGS

All entries are supposed to be reproducible either on a typesetter or on a terminal. However, on a terminal some information is necessarily lost.

Pages bearing the same name in both manuals will result in the *UniPlus*+ *Administrator's Manual* entry being printed first, if no *section* argument is supplied.

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

Mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

EXAMPLE

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

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

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

NAME

mkdir — make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777 (possibly altered by *umask*(1)). Standard entries, ., for the directory itself, and .., for its parent, are made automatically. These and other directories beginning with . are not visible in listings unless you use the **-a** option to *ls*.

Mkdir requires write permission in the parent directory.

EXAMPLE

```
mkdir letters
```

creates a directory **letters** as a subdirectory of the directory you are in at the time you employ the command.

SEE ALSO

rm(1), sh(1), umask(1).

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

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. Its use 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 new-line character and a null character; the null character terminates the message so it can be easily used when retrieved, the new-line 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;

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) | | 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.

EXAMPLE

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 contains the extracted error messages.

SEE ALSO

lseek(2).

BUGS

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

AUTHORS

Bill Joy and Charles Haley.

NAME

mm, osdd, checkmm − print/check documents formatted with the MM macros

SYNOPSIS

mm [options] [files]

osdd [options] [files]

checkmm [files]

DESCRIPTION

Mm can be used to type out documents using *nroff* and the MM text-formatting macro package. It has options to specify preprocessing by *tbl*(1) and/or *neqn* (see *eqn*(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 selected.

Osdd is equivalent to the command **mm −mosd**. For more information about the OSDD adapter macro package, see *mosd*(5).

Options for *mm* are given below. Any other arguments or flags (e.g., **−rC3**) are passed to *nroff* or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

−T*term* Specifies the type of output terminal; for a list of recognized values for *term*, type **help term2**. If this option is *not* used, *mm* will use the value of the shell variable **$TERM** from the environment (see *profile*(4) and *environ*(5)) as the value of *term*, if **$TERM** is set; otherwise, *mm* will use **450** as the value of *term*. If several terminal types are specified, 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**, and **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*.

−y Causes *mm* to use the non-compacted version of the macros (see *mm*(5)).

Checkmm is a program for checking the contents of the named *files* for errors in the use of the Memorandum Macros, missing or unbalanced *neqn* delimiters, and **.EQ/.EN** pairs. Note: The user need not use the *checkeq* program (see *eqn*(1)). Appropriate messages are produced. The program skips all directories, and if no file name is given, standard input is read.

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 ghh*
tbl ghh* | nroff −cm −T450−12 −h −rC3
```

Mm reads the standard input when − is specified instead of any file names. (Mentioning other files together with − leads to disaster.) This option allows *mm* to be used as a filter, e.g.:

```
cat dws | 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 line-feed (rather than return or new-line) 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 above).
4. If you lie to *mm* about the kind of terminal its output will be printed on, you'll 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.

SEE ALSO

col(1), cw(1), env(1), eqn(1), greek(1), mmt(1), nroff(1), tbl(1), profile(4), mm(5), mosd(5), term(5).

DIAGNOSTICS

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

checkmm "Cannot open *filename*" if file(s) is unreadable. The remaining output of the program is diagnostic of the source file.

NAME

mmt, mvt − typeset documents, view graphs, and slides

SYNOPSIS

mmt [options] [files]

mvt [options] [files]

DESCRIPTION

These two commands are very similar to *mm*(1), except that they both typeset their input via *troff*(1), as opposed to formatting it via *nroff*; *mmt* uses the MM macro package, while *mvt* uses the Macro Package for View Graphs and Slides. These two commands have options to specify preprocessing by *tbl*(1) and/or *eqn*(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.

Options are given below. Any other arguments or flags (e.g., **−rC3**) are passed to *troff*(1) or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, these commands print a list of their options.

- **−e** Causes these commands to invoke *eqn*(1); also causes *eqn* to read the **/usr/pub/eqnchar** file (see *eqnchar*(5)).
- **−t** Causes these commands to invoke *tbl*(1).
- **−Tst** Directs the output to the MH STARE facility.
- **−a** Invokes the **−a** option of *troff*(1).
- **−y** Causes *mmt* to use the non-compacted version of the macros (see *mm*(5)). No effect for *mvt*.

These commands read the standard input when − is specified instead of any file names.

Mvt is just a link to *mmt*.

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**, 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*.

EXAMPLE

```
mmt -t -rC3 -12 -Tst file
```

is equivalent to

```
tbl file | troff -cm -Tst -12 -h -rC3
```

SEE ALSO

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

DIAGNOSTICS

"m[mv]t: no input file" if none of the arguments is a readable file and the command is not used as a filter.

NAME

more − file perusal filter for crt viewing

SYNOPSIS

more [−**dfl***n*] [+*linenumber* | +/*pattern*] [name ...]

DESCRIPTION

More is a filter which allows examination of a 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.

If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. If a space is preceded by an integer, that number of lines is printed. If the user hits **d** or control-D, 11 more lines (usually half a screenful) are displayed (a "scroll").

More looks in the user's environment and, if necessary, 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.

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 options are available:

−**n** is an integer which is the size (in lines) of the window which *more* will use instead of the default.

−**c** *More* will 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 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, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters 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 fold lines erroneously.

−**l** causes *more* not to treat control-L (form feed) specially. If this 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** Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

−**u** Normally, *more* 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 stand-out mode for

underlined information in the source file. The **−u** option suppresses this processing.

+ *linenumber*
option causes *more* to start up at *linenumber*

+ */pattern*
causes *more* to start up two lines before the line containing the regular expression *pattern.*

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) :

= display the current line number

v start up the editor *vi* at the current line

h help command; give a description of all the *more* commands

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

i:p skip 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 is rung and nothing else happens.

i:f display the current file name and line number.

i:q or :Q
exit from *more* (same as q or Q).

. (dot) repeat the previous command.

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

*i*s skip *i* lines and print a screenful of lines

*i*f skip *i* screenfuls and print a screenful of lines

*i*n skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense)

*i*p skip 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 is rung and nothing else happens.

q or Q
Exit from more.

i/expr
search for the *i*-th occurrence of the regular expression *expr.* If there are less than *i* occurrences of *expr* and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place 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.

' (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
invoke a shell with *command.*

The commands take effect immediately, i.e., it is not necessary to type a carriage return. 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.

At any time when output is being sent to the terminal, the user can hit 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 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.

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

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

EXAMPLE

```
nroff −ms +2 doc.n | more
```

would show the *nroff* output on the terminal screen.

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

AUTHOR

Eric Shienbrood

NAME

newform − change the format of a text file

SYNOPSIS

newform [−s] [−itabspec] [−otabspec] [−bn] [−en] [−pn] [−an] [−f] [−cchar] [−ln] [files]

DESCRIPTION

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

Except for −s, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like −e 15 −l60 will yield results different from −l 60 −e15. Options are applied to all *files* on the command line.

−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 file-name
```

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 file-name

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

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 – log in to a new group

SYNOPSIS

newgrp [–] [group]

DESCRIPTION

Newgrp changes the group identification of its caller, analogously to *login*(1). The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

Newgrp without an argument changes the group identification to the group in the password file; in effect it changes the group identification back to the caller's original group.

An initial – flag causes the environment to be changed to the one that would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user himself 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.

When most users log in, they are members of the group named **other**.

EXAMPLE

```
newgrp grpnam
```

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

FILES

/etc/group
/etc/passwd

SEE ALSO

login(1), group(4).

BUGS

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

NAME

news − print news items

SYNOPSIS

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

DESCRIPTION

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

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

The **−a** option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

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

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

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

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

EXAMPLE

```
news
```

will print out all files in **/usr/news** that have not been read previously by the account owner.

FILES

/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 super-user may run commands with priority higher than normal by using a negative increment, e.g., **−−10**.

EXAMPLE

For the Bourne shell:

```
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, the same is achieved by typing in

```
nice +10 date
```

SEE ALSO

nohup(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 [−**h**type] [−**b**type] [−**f**type] [−**v**start#] [−**i**incr] [−**p**] [−**l**num] [−**s**sep] [−**w**width] [−**n**format] [−**d**delim] file

DESCRIPTION

Nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

Nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

−b *type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: **a**, number all lines; **t**, number lines with printable text only; **n**, no line numbering; **p** *string*, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is **t** (text lines numbered).

−h *type* Same as **−b** *type* except for header. Default *type* for logical page header is **n** (no lines numbered).

−f *type* Same as **−b** *type* except for footer. Default for logical page footer is **n** (no lines numbered).

−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.

EXAMPLE

```
nl −v10 −i10 −d!+ file1 file2
```

will number "file1" and "file2" starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

NAME

nm − print name list

SYNOPSIS

nm [**−gnoprsu**] [file ...]

DESCRIPTION

Nm prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in **a.out** are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

−g Print only global (external) symbols.

−n Sort numerically rather than alphabetically.

−o Prepend file or archive element name to each output line rather than only once. This option can be used to make piping to *grep*(1) more meaningful.

−p Don't sort; print in symbol-table order.

−r Sort in reverse order.

−s Sort according to the size of the external symbol (computed from the difference between the value of the symbol and the value of the symbol with the next highest value). This difference is the value printed. This flag turns on **−g** and **−n** and turns off **−u** and **−p**.

−u Print only undefined symbols.

EXAMPLE

```
nm
```

prints the symbol list of **a.out** , the default output file for the C compiler.

SEE ALSO

ar(1), a.out(5), ar(5).

NAME

nohup – run a command immune to hangups (*sh* only)

SYNOPSIS

nohup command [arguments]

DESCRIPTION

Nohup executes *command* immune to terminate (EOT, control-D) signal from the controlling terminal. With *nohup*, the priority is automatically incremented by 5. *Nohup* should be used with processes running in background (with "&") in order to prevent it from responding to interrupts or stealing the input from the next person who logs in on the same terminal. In *csh*, processes run in background are automatically immune to hangups.

If output is not redirected by the user, it will be sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**.

EXAMPLE

```
nohup nroff -ms docsfile | lpr
```

runs the *nroff* command shown, immune to hangups, quits, and interrupts.

FILES

nohup.out standard output and standard error file.

SEE ALSO

csh(1), nice(1), nice(2).

NAME

nroff − format text

SYNOPSIS

nroff [options] [files]

DESCRIPTION

Nroff formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers. Its capabilities are described in the *NROFF/TROFF User's Manual* cited below.

An argument consisting of a minus (−) is taken to be a file name corresponding to the standard input. The *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 line-feed or new-line (new-lines 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* 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 non-compacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.

−c *name* Prepend to the input *files* the compacted macro files **/usr/lib/macros/cmp.[nt].[dt].***name* and **/usr/lib/macros/ucmp.[nt].***name*.

−k *name* Compact the macros used in this invocation of *nroff*, placing the output in files **[dt].***name* in the current directory.

−T *name* Prepare output for specified terminal. Known *name*s are **37** for the (default) TELETYPE® Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300s** for the DASI 300s, **300** for the DASI 300, **450** for the DASI 450, **lp** for a (generic) ASCII line printer, **382** for the DTC-382, **4000A** for the Trendata 4000A, **832** for the Anderson Jacobson 832, **X** for a (generic) EBCDIC printer, and **2631** for the Hewlett Packard 2631 line printer.

−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.

EXAMPLE

nroff -o4,8-10 -T300S -mabc file1 file2

requests formatting of pages 4, 8, 9, and 10 of a document contained in the files named "file1" and "file2", specifies the output terminal as a DASI-300S, and invokes the macro package *abc*.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*

SEE ALSO

col(1), cw(1), eqn(1), greek(1), mm(1), tbl(1), troff(1), mm(5).
NROFF/TROFF User's Manual
A TROFF Tutorial

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 *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

nroff7 − text formatting and typesetting

SYNOPSIS

nroff7 [option] ... [file] ...

DESCRIPTION

Nroff7 formats text in the named *files* for typewriter-like devices. See also *nroff*(1) *troff*(1), and *troff7*(1). The full capabilities of *nroff* and *troff* are described in the *Nroff/Troff User's Manual.*

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (−) is taken to be a file name corresponding to the standard input.

The 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.

−n*N* Number first generated page *N*.

−s*N* Stop every *N* pages. *Nroff7* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline.

−m*name* Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

−r*aN* Set register *a* (one-character) to *N*.

−i Read standard input after the input files are exhausted.

−q Invoke the simultaneous input-output mode of the **rd** request.

−T*name* Prepare output for specified terminal. Known *names* are **37** for the (default) Teletype Corporation Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300S** for the DASI-300S, **300** for the DASI-300, and **450** for the DASI-450 (Diablo Hyterm).

−e Produce equally-spaced words in adjusted lines, using full terminal resolution.

−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.

EXAMPLE

```
nroff7 -s4 -me filea
```

will *nroff7* the named file using the **−me** macro package, stopping every 4 pages.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff7*

NAME

od − octal dump

SYNOPSIS

od [**−bcdosx**] [file] [[+]offset[.][**b**]]

DESCRIPTION

Od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, **−o** is default. The meanings of the format options are:

−b Interpret bytes in octal.

−c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=**\0**, backspace=**\b**, form-feed=**\f**, new-line=**\n**, return=**\r**, tab=**\t**; others appear as 3-digit octal numbers.

−d Interpret words in unsigned decimal.

−o Interpret words in octal.

−s Interpret 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.

EXAMPLE

```
od -d filea +2
```

produces an octal dump of "filea" divided up into 32-bit words expressed in decimal equivalents with the dump starting point offset by 2 octal bytes.

SEE ALSO

dump(1).

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] 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*. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

Pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;
the file name has more than 12 characters;
the file has links;
the file is a directory;
the file cannot be opened;
no disk storage blocks will be saved by packing;
a file called *name*.**z** already exists;
the **.z** file cannot be created;
an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended **.z** extension. Directories cannot be compressed.

Pcat does for packed files what *cat*(1) does for ordinary files. 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.

EXAMPLE

pack file1

will pack file "file1" into "file1.z" and removes "file1" if packing is successful.

NAME

passwd – change login password

SYNOPSIS

passwd name

DESCRIPTION

This command changes (or installs) a password associated with the *login name*.

The program prompts for the old password (if any) and then for the new one (twice). The caller must supply these. New passwords should be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. Only the first eight characters of the password are significant.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password. Only the super-user can create a null password.

The password file is not changed if the new password is the same as the old password, or if the password has not "aged" sufficiently; see *passwd*(4).

EXAMPLE

passwd

will give the respondence

Changing password for <*username*>

and will then prompt for your present password and for the new password (twice).

FILES

/etc/passwd

SEE ALSO

login(1), crypt(3C), passwd(4).

NAME

paste − merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste **−d** list file1 file2 ...
paste **−s** [**−d** list] file1 file2 ...

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* subsumes the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if − is used in place of a file name.

The meanings of the options are:

−d Without this option, the new-line characters of each but the last file (or last line in case of the **−s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).

list One or more characters immediately following **−d** replace the default *tab* as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no **−s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), **** (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use *−d*"\\\\").

−s Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **−d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.

− May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLE

```
ls | paste −d" " −
```

list directory in one column.

```
ls | paste − − − −
```

list directory in four columns.

```
paste −s −d"\t\n" file
```

combine pairs of lines into lines.

SEE ALSO

cut(1), grep(1),
pr(1): **pr −t −m**... works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

DIAGNOSTICS

line too long Output lines are restricted to 511 characters.

too many files Except for −s option, no more than 12 input files may be specified.

NAME

pr − print files

SYNOPSIS

pr [options] [files]

DESCRIPTION

Pr prints the named files on the standard output. If *file* is −, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **−s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing.

The below *options* may appear singly or be combined in any order:

+*k* Begin printing with page *k* (default is 1).

−*k* Produce *k*-column output (default is 1). The options **−e** and **−i** are assumed for multi-column output. Also, the **−k** option *must* be used if the **−w** (column width) option is used.

−a Print multi-column output across the page.

−m Merge and print all files simultaneously, one per column (overrides the −*k*, and **−a** options).

−d Double-space the output.

−e *ck* Expand *input* tabs to character positions *k*+1, 2* *k*+1, 3* *k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).

−i *ck* In *output*, replace white space wherever possible by inserting tabs to character positions *k*+1, 2* *k*+1, 3* *k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

−n *ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **−m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

−w *k* For multi-column output, set the width of a line to *k* character positions instead of the default 72 characters. This option *must* be used with the **−k** (number of columns) 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 to be printed instead of the file name.

-p Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

-f Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

-r Print no diagnostic reports on failure to open files.

-t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

-s *c* Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

EXAMPLE

```
pr -3dh "file list" file1 file2
```

prints "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

/dev/tty* to suspend messages

SEE ALSO

cat(1).

NAME

printenv − print out the environment

SYNOPSIS

printenv [argument]

DESCRIPTION

Printenv takes an environment variable name as an argument and prints only the value of that variable. If no argument is given, it prints the values for the entire environment.

Examples of environment variable names are:

HOME	path name of user's home directory.
SHELL	the shell present at login.
PATH	search path for binary programs.
TERM	type of terminal used.
USER	the login name of the user.
TERMCAP	terminal capabilities string.
EXINIT	a startup list of commands read by *ex*, *edit* and *vi*.

EXAMPLE

```
printenv HOME
```

prints the path name of your home directory.

SEE ALSO

csh(1), sh(1), environ(4).

NAME

prof − display profile data

SYNOPSIS

prof [−**v**] [−**a**] [−**l**] [−low [−high]] [file]

DESCRIPTION

Prof interprets the file **mon.out** produced by the *monitor*(3C) subroutine. Under default modes, the symbol table in the named object file (**a.out** default) is read and correlated with the **mon.out** profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

If the −**a** option is used, all symbols are reported rather than just external symbols. If the −**l** option is used, the output is listed by symbol value rather than decreasing percentage.

If the −**v** option is used, all printing is suppressed and a graphic version of the profile is produced on the standard output for display by the *tplot*(1G) filters. The optional arguments *low* and *high*, by default 0 and 100, cause a selected percentage of the profile to be plotted with accordingly higher resolution.

In order for the number of calls to a routine to be tallied, the −**p** option of *cc* must have been given when the file containing the routine was compiled. This option also arranges for the **mon.out** file to be produced automatically.

EXAMPLE

If **a.out** has been compiled with the −**p** option and has been executed, then

```
prof a.out
```

would print profile information for each routine in **a.out**.

FILES

mon.out	for profile
a.out	for namelist

SEE ALSO

cc(1), tplot(1G), profil(2), monitor(3C).

BUGS

Beware of quantization errors.

NAME

prs − print an SCCS file

SYNOPSIS

prs [−**d**[dataspec]] [−**r**[SID]] [−**e**] [−**l**] [−**a**] files

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(4)) in a user supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to *prs*, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

−**d**[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.

−**r**[*SID*] Used to specify the *S*CCS *ID*entification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

−**e** Requests information for all deltas created *earlier* than and including the delta designated via the −**r** keyletter.

−**l** Requests information for all deltas created *later* than and including the delta designated via the −**r** keyletter.

−**a** Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(1)) and existing, i.e., delta type = *D*, deltas. If the −**a** keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(4)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by **\t** and carriage return/new-line is specified by **\n**.

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
: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:

EXAMPLE

prs −d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

Users and/or user IDs for s.file are:
xyz
131
abc

prs −d"Newest delta for pgm :M:: :I: Created :D: By :P:" −r s.file

may produce on the standard output:
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a *special case*:

prs s.file

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
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

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(4).
"Source Code Control System User's Guide"

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps – report process status

SYNOPSIS

ps [options]

DESCRIPTION

Ps prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. Otherwise, the information that is displayed is controlled by the following *options*:

- **–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. (Normally, a short listing containing only process ID, terminal ("tty") identifier, cumulative execution time, and the command name is printed.) See below for meaning of columns in a full listing.
- **–l** Generate a *long* listing. See below.
- **–c** *corefile* Use the file *corefile* in place of **/dev/mem**.
- **–s** *swapdev* Use the file *swapdev* in place of **/dev/swap**. This is useful when examining a *corefile*; a *swapdev* of **/dev/null** will cause the user block to be zeroed out.
- **–n** *namelist* The argument will be taken as the name of an alternate *namelist* (**/unix** is the default).
- **–t** *tlist* Restrict listing to data about the processes associated with the terminals given in *tlist*, where *tlist* can 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 comma and/or one or more spaces.
- **–p** *plist* Restrict listing to data about processes whose process ID numbers are given in *plist*, where *plist* is in the same format as *tlist*.
- **–u** *ulist* Restrict listing to data about processes whose user ID numbers or login names are given in *ulist*, where *ulist* is in the same format as *tlist*. In the listing, the numerical user ID will be printed unless the **–f** option is used, in which case the login name will be printed.
- **–g** *glist* Restrict listing to data about processes whose process groups are given in *glist*, where *glist* is a list of process group leaders and is in the same format as *tlist*.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (*full* or *long*) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options only determine what information is provided for a process; they do *not* determine which processes will be listed.

F (l) Flags (octal and additive) associated with the process:
- 01 in core;
- 02 system process;

04 locked in core (e.g., for physical I/O);
10 being swapped;
20 being traced by another process;
40 another tracing flag.

S (l) The state of the process:
0 non-existent;
S sleeping;
W waiting;
R running;
I intermediate;
Z terminated;
T stopped;
X growing.

UID (f,l) The user ID number of the process owner; the login name is printed under the **−f** option.

PID (all) The process ID of the process; 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.

STIME (f) Starting time of the process.

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 process, if resident; otherwise, the disk address.

SZ (l) The size in blocks 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.

TTY (all) The controlling terminal for the process.

TIME (all) The cumulative execution time for the process.

CMD (all) The command name; the full command name and its arguments are printed under the **−f** option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Under the **−f** option, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the **−f** option, is printed in square brackets.

EXAMPLE

```
ps −ef
```

displays information about all processes, with or without terminals.

FILES

/unix	system namelist.
/dev/mem	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

kill(1), nice(1).

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

NAME

ptx − permuted index

SYNOPSIS

ptx [options] [input [output]]

DESCRIPTION

Ptx generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of each line. *Ptx* output is in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where **.xx** is assumed to be an *nroff* or *troff* macro provided by the user, or provided by the *mptx*(5) macro package. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The following *options* can be applied:

−f Fold upper and lower case letters for sorting.

−t Prepare the output for the phototypesetter.

−w *n* Use the next argument, *n*, as the length of the output line. The default line length is 72 characters for *nroff* and 100 for *troff*.

−g *n* Use the next argument, *n*, as the number of characters that *ptx* will reserve in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3.

−o *only* Use as keywords only the words given in the *only* file.

−i *ignore* Do not use as keywords any words given in the *ignore* file. If the **−i** and **−o** options are missing, use **/usr/lib/eign** as the *ignore* file.

−b *break* Use the characters in the *break* file to separate words. Tab, new-line, and space characters are *always* used as break characters.

−r Take any leading non-blank characters of each input line to be a reference identifier (as to a page or chapter), separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx*.

EXAMPLE

If "file1" contains:

```
once upon a time
in the middle of a large
dark forest
```

```
ptx file1
```

responds with:

```
.xx "" "" "dark forest" ""
.xx "" "dark" "forest" ""
.xx "" "in the middle of a" "large" ""
.xx "" "in the" "middle of a large" ""
.xx "" "" "once upon a time" ""
.xx "" "once" "upon a time" ""
```

FILES

/bin/sort
/usr/lib/eign
/usr/lib/tmac/tmac.ptx

SEE ALSO

nroff(1), troff(1), mm(5), mptx(5).

BUGS

Line length counts do not account for overstriking or proportional spacing. Lines that contain tildes (~) are botched, because *ptx* uses that character internally.

NAME

put − puts a file onto a remote machine.

SYNOPSIS

put [**−p** *port*] [−sSPEED] [**−i** [ID]] *fromfile* [*tofile*]
put [**−p** *port*] [−sSPEED] **−c** *command* [*args*] ...

DESCRIPTION

Put is part of system of programs useful for transferring files between UNIX systems. It is the "uploader" designed to transmit files from a local machine to a remote machine. For a brief discussion of the take/put system and installation instructions, see the companion document: *Installation and Overview of the UniSoft Take/Put File Transfer System.*

The default port is **/dev/tty0**; the **−p** option can be used to specify an alternate output port. The default speed is determined by the system; the **−s** option can be used to specify a speed. If *tofile* is unspecified, then it is assumed to be the same as *fromfile*. If *fromfile* is a directory, *tofile* must be a directory on the remote machine (or if nonexistent, the last existent directory specified in the pathname must be writable).

The **−i[ID]** option specifies a system ID and is the mechanism for remapping pathnames on the remote machine. The system ID is passed to the remote machine where it is used to generate pathname prefixes (using the **/etc/take_oem** file) which are appended to the *tofile* pathname supplied by *put*. If an ID is specified when using the **−i** option, it is used on the remote machine. If no ID is specified, the default ID is read from the **/etc/sys_id** file if it exists; if the **/etc/sys_id** does not exist, the system ID is considered to be the user name of the invoker of *put7* (i.e., the user who logged in over the port used).

The **−c** option is useful for executing an arbitrary command on the remote machine. All arguments following the **−c** flag are collected, transmitted to the remote machine and executed as a single command. The standard input to the *put* program is sent to the remote machine to become the standard input to the command specified. The standard error of the remote command becomes the standard error of *put*. The standard output of the remote command is not returned. The exit status of the remote command is returned as the exit status of *put*.

In order to perform its function, *put*(1C) interfaces with the program **/usr/bin/put7** on the remote machine.

EXAMPLE

```
put /a/b/c
```

puts the contents of the directory (or file) "/a/b/c" on the local machine into a similarly named directory (or file) on the remote machine; if "/a/b/c" did not previously exist on the remote machine, it is created; otherwise it is overwritten.

```
put file.c /x/y/z
```

puts the contents of "file.c" on the local machine into "/x/y/z/file.c" on the remote machine. Note that "file.c" is created on the remote machine if "z" is a directory; if "z" is a file rather than a directory, its contents are overwritten but its name remains "z" rather than becoming "file.c".

FILES

fromfile The local file name. When using the **-i** option, this file should be specified as a pathname starting at the root of the local machine.

tofile The remote file name; if *tofile* is null, *tofile* is defaulted to *fromfile*.

SEE ALSO

cu(1C), take(1C)
Installation and Overview of the UniSoft Take/Put File Transfer System

NAME

put7 − puts a file onto a remote machine.

SYNOPSIS

put7 [**−p** *port*] [**−sSPEED**] [**−i[ID]**] *fromfile* [*tofile*]
put7 [**−p** *port*] [**−sSPEED**] **−c** *command* [*args*] ...

DESCRIPTION

Put7 is part of system of programs useful for transferring files between UNIX systems. It is the "uploader" designed to transmit files from a local machine to a remote machine. For a brief discussion of the take/put system and installation instructions, see the companion document: "*Overview of the UniSoft Take/Put File Transfer System*".

The default port is **/dev/tty0**; the **−p** option can be used to specify an alternate output port. The default speed is determined by the system; the **−s** option can be used to specify a speed. If *tofile* is unspecified, then it is assumed to be the same as *fromfile*. If *fromfile* is a directory, *tofile* must be a directory on the remote machine (or if nonexistent, the last existent directory specified in the pathname must be writable).

The **−i[ID]** option specifies a system ID and is the mechanism for remapping pathnames on the remote machine. The system ID is passed to the remote machine where it is used to generate pathname prefixes (using the **/etc/take_oem** file) which are appended to the *tofile* pathname supplied by *put7*. If an ID is specified when using the **−i** option, it is used on the remote machine. If no ID is specified, the default ID is read from the **/etc/sys_id** file if it exists; if the **/etc/sys_id** does not exist, the system ID is considered to be the user name of the invoker of *put6* (i.e., the user who logged in over the port used).

The **−c** option is useful for executing an arbitrary command on the remote machine. All arguments following the **−c** flag are collected, transmitted to the remote machine and executed as a single command. The standard input to the *put7* program is sent to the remote machine to become the standard input to the command specified. The standard error of the remote command becomes the standard error of *put7*. The standard output of the remote command is not returned. The exit status of the remote command is returned as the exit status of *put7*.

In order to perform its function, *put7*(1C) interfaces with the program **/usr/bin/put6** on the remote machine.

EXAMPLE

```
put7 /a/b/c
```

puts the contents of the directory (or file) "/a/b/c" on the local machine into a similarly named directory (or file) on the remote machine; if "/a/b/c" did not previously exist on the remote machine, it is created; otherwise it is overwritten.

```
put7 file.c /x/y/z
```

puts the contents of "file.c" on the local machine into "/x/y/z/file.c" on the remote machine. Note that "file.c" is created on the remote machine if "z" is a directory; if "z" is a file rather than a directory, its contents are overwritten but its name remains "z" rather than becoming "file.c".

FILES

fromfile The local file name. When using the **-i** option, this file should be specified as a pathname starting at the root of the local machine.

file The remote file name; if *tofile* is null, *tofile* is defaulted to *fromfile*.

SEE ALSO

cu(1C), take7(1)
Overview of the UniSoft Take/Put File Transfer System

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the path name of the working (current) directory.

EXAMPLE

```
pwd
```

produces a pathname, such as **/usr/games**, indicating what directory you are currently in.

SEE ALSO

cd(1).

DIAGNOSTICS

"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred to a UNIX system programming counselor.

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 file name of the form "rhost:path", or a local file name (containing no ':' characters, or a '/' before any ':'s.)

If the **−r** 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.

If *path* is not a full path name, it is interpreted relative to your 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; your 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. Hostnames may also take the form "rhost.rname" to use *rname* rather than the current user name on the remote host.

SEE ALSO

remsh(1N), rlogin(1N).

BUGS

Doesn't detect in all cases the fact that a target of a copy might be a file in cases where only a directory should be legal.

This command is provisional and may be changed in future releases.

NAME

rcvhex − translates Motorola S-records from downloading into a file

SYNOPSIS

rcvhex [**−p** port] [**−c** command] file

DESCRIPTION

Rcvhex translates Motorola S-records shipped from a port into a file. The following 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.

AUTHOR

Asa Romberger, UniSoft Systems

NAME

regcmp − regular expression compile

SYNOPSIS

regcmp [−] files

DESCRIPTION

Regcmp, in most cases, precludes the need for calling *regcmp*(3X) from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file*.**i**. If the − option is used, the output will be placed in *file*.**c**. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *File*.**i** files may thus be *included* into C programs, or *file*.**c** files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex*(*abc*,*line*) will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

EXAMPLE

```
name    "([A-Za-z][A-Za-z0-9_]*)$0"
telno   "\({0,1}([2-9][01][1-9])$0\){0,1} *"
        "([2-9][0-9]{2})$1[ -]{0,1}"
        "([0-9]{4})$2"
```

In the C program that uses the *regcmp* output,

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

will apply the regular expression named *telno* to *line*.

SEE ALSO

regcmp(3X).

NAME

remsh − remote shell

SYNOPSIS

remsh host [−**l** username] [−**n**] command
host [−**l** username] [−**n**] command

DESCRIPTION

Remsh connects to the specified *host*, and executes the specified *command*. *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 *username* used is the same as your local username, unless you specify a different remote name with the −**l** option. This remote name must be equivalent (in the sense of *rlogin* (1N)) to the originating account; no provision is made for specifying a password with a command.

If you omit *command*, then instead of executing a single command, you will be logged in on the remote host using *rlogin* (1N).

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

remsh otherhost cat remotefile >> localfile

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

remsh otherhost cat remotefile ">>" otherremotefile

appends "remotefile" to "otherremotefile".

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 host names for local machines maybe linked to the *remsh* command in some convenient place, normally in the directory **/usr/host**. If this directory is in one's search path, then the *remsh* can be omitted. If no input is desired, you should redirect the input of *remsh* to **/dev/null** using the −**n** option.

FILES

/etc/hosts
/usr/hosts/*
/etc/remsh

SEE ALSO

rlogin (1N).

BUGS

You cannot run an interactive command (like *vi* (1)); use *rlogin* (1).

This command is provisional and may change in future releases.

NAME

reset − reset the teletype bits to a sensible state

SYNOPSIS

reset

DESCRIPTION

Reset sets the terminal to cooked mode, turns off "cbreak" and "raw" modes, turns on "nl", and restores special characters that are undefined to their default values.

This is most useful after a program dies leaving a terminal in a funny state; you have to type **<LF>reset<LF>** to get it to work as <CR> often doesn't work; often none of this will echo.

It isn't a bad idea to follow *reset* with *tset*(1).

EXAMPLE

```
reset
```

returns the user's terminal to a usable state after being accidentally set by an interrupted process.

SEE ALSO

stty(1), tset(1).

BUGS

Doesn't set tabs properly; it can't intuit personal choices for interrupt and line kill characters, so it leaves these the old UNIX standards ^? (delete) for interrupt and @ for line kill.

It could well be 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

rlogin − remote login

SYNOPSIS

rlogin rhost [−**e** *c*] [−**l** username]
rhost [−**l** username]

DESCRIPTION

Rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost.*

Each host has a file **/etc/hosts.equiv** which contains a list of *rhosts* with which it shares account names. (The host names must be the standard names as described in *remsh* (1N) and printed by *login* (1).) 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).

All echoing takes place at the remote site, so that (except for delays) the *rlogin* is transparent. Flow control via control-S (^S) and control-Q (^Q) is handled properly. A line of the form "~." disconnects from the remote host, where "~" is the escape character. A different escape character may be specified by the −**e** option. Other *cu* (1C) "~" options available; see *cu* (1C) documentation for details.

SEE ALSO

cu(1C), remsh(1N).

FILES

/usr/hosts/* for *rhost* version of the command

BUGS

The "~%put" *cu* function should be made to work.

More terminal characteristics should be propagated.

This command is provisional and may be revised and/or renamed in future releases.

NAME

rm, rmdir − remove files or directories

SYNOPSIS

rm [**−fri**] file ...

rmdir dir ...

DESCRIPTION

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

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with **y** the file is deleted, otherwise the file remains. No questions are asked when the **−f** option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument **−r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **−i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **−r**, whether to examine each directory.

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

EXAMPLE

```
rm -r dirname
```

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

SEE ALSO

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 .*

NAME

rmdel – remove a delta from an SCCS file

SYNOPSIS

rmdel –r*SID* files

DESCRIPTION

Rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the 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 path name does not begin with **s.**) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are either: (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

EXAMPLE

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

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

FILES

x-file (see *delta*(1))
z-file (see *delta*(1))

SEE ALSO

delta(1), get(1), help(1), prs(1), sccsfile(4).
Source Code Control System User's Guide

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

rstat − network statistics program

SYNOPSIS

rstat [−**Amisr**] [−**p**protocol] [−**a**] [interval] [system] [core]

DESCRIPTION

The *rstat* command symbolically displays the contents of various network-related data structures. The options have the following meaning:

−a show the state of all sockets; this is the default

−i show the state of interfaces which have been auto-configured

−m show statistics recorded by the memory management routines (the network manages a "private share" of memory)

−p*proto*
show the state of sockets utilizing protocol *proto*; the protocol is specified symbolically, e.g., "tcp"

−s show per-protocol statistics

−r show the routing tables

−A give the kernel address of the protocol "state block" associated with an active socket (used for debugging)

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

If an *interval* is specified, *rstat* will continuously display the requested information, pausing *interval* seconds before refreshing the screen.

DISPLAYS

There are a number of display formats, depending on the information presented. The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and, optionally, the internal state of the protocol.

Address formats vary according to their "address family". Internet address are displayed as "address/port", where port is printed symbolically if it is a well-known service (e.g., telnet). The address portion is a hex representation in the "standard network format". Unspecified, or "wildcard", addresses and ports appear as "*". Raw socket addresses may appear unspecified (e.g., "unspec") if no address was supplied when the socket was created.

Protocols are normally printed symbolically, though they may also appear as "protocol-family/protocol".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network address (currently Internet specific) 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"), and whether the route is a direct route ("D"). Direct routes are created for each interface attached to the local host. 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 then discard it. 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.

BUGS

The formats and all need to be redone. Network address should be displayed symbolically (e.g., "ucbmonet", "sri-prmh"). Interval statistics are more convenient when watching the net during a transfer. The notion of errors is ill-defined.

NAME

ruptime − show host status of local machines

SYNOPSIS

ruptime [**−a**]

DESCRIPTION

Ruptime gives a status line like *uptime* for each machine on the 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.

FILES

/etc/whod.* data files

SEE ALSO

rwho(1N).

BUGS

This command is provisional and may change in future releases.

NAME

rwho − who is logged in on local machines

SYNOPSIS

rwho [**−a**] [**−u**] [*systemname(s)*] [− *systemname(s)*]

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 an hour or more, then the user will be omitted from the output of *rwho* unless the **−a** flag is given. *Rwho* normally sorts its output by *systemname*, the **−u** option will cause *rwho* to sort its output by *username*. If a *systemname* is given, only information for that system in printed. If a *−systemname* is given, output is suppressed for that system.

FILES

/etc/whod.* information about other machines

BUGS

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

This command is provisional and may change in future releases.

NAME

sact − print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

Sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the **−e** option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of − is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

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

Field 1 specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.

Field 2 specifies the SID for the new delta to be created.

Field 3 contains the logname of the user who will make the delta (i.e., executed a *get* for editing).

Field 4 contains the date that **get −e** was executed.

Field 5 contains the time that **get −e** was executed.

EXAMPLE

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 eryk 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 "eryk". The **get −e** for the file was done on 82/11/10 at 16:10:35.

SEE ALSO

delta(1), get(1), unget(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sadp – disk access profiler

SYNOPSIS

sadp [**−th**] [**−d** device[−drive]] s [n]

DESCRIPTION

Sadp reports disk access location and seek distance, in tabular or histogram form. It samples disk activity once every second during an interval of *s* seconds. This is done repeatedly if *n* is specified. Cylinder usage and disk distance are recorded in units of eight cylinders.

Valid values of *device* are **rp06**, **rm05**, and **disk**. *Drive* specifies the disk drives and it may be:

a drive number in the range supported by *device*,
two numbers separated by a minus (indicating an inclusive range),

or

a list of drive numbers separated by commas.

Up to eight disk drives may be reported. The **−d** option may be omitted, if only one *device* is present.

The **−t** flag causes the data to be reported in tabular form. The **−h** flag produces a histogram on the printer of the data. Default is **−t**.

EXAMPLE

```
sadp −d rp06 −0 900 4
```

will generate 4 tabular reports, each describing cylinder usage and seek distance of **rp06** disk drive 0 during a 15 minute interval.

FILES

/dev/kmem

NAME

sag − system activity graph

SYNOPSIS

sag [options]

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 *options* are passed thru to *sar*:

−s *time* Select data later than *time* in the form hh[:mm]. Default is 08:00.

−e *time* Select data up to *time*. Default is 18:00.

−i *sec* Select data at intervals as close as possible to *sec* seconds.

−f *file* Use *file* as the data source for *sar*. Default is the current daily data file **/usr/adm/sa/sa***dd*.

Other *options*:

−T *term* Produce output suitable for terminal *term*. See *tplot*(1G) for known terminals. 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 (A/(A+B))*100, and A + B / C + D is (A+B)/(C+D). *Lo* and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

A single *spec* is permitted for the x axis. If unspecified, *time* is used. Up to 5 *spec*'s separated by ; may be given for **−y**. Enclose the **−x** and **−y** arguments in "" if blanks or \<CR> are included. The **−y** default is:

−y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"

EXAMPLE

```
sag
```

wil show today's CPU utilization.

FILES

/usr/adm/sa/sa*dd* daily data file for day *dd*.

SEE ALSO

sar(1), tplot(1G).

NAME

sar – system activity reporter

SYNOPSIS

sar [–**ubdycwaqvmA**] [–**o** file] t [n]

sar [–**ubdycwaqvmA**] [–**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** 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 –**f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa** *dd* for the current day *dd*. The starting and ending times of the report can be bounded via the –**s** and –**e** *time* arguments of the form *hh*[:*mm*[:*ss*]]. The –**i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

–**u** Report CPU utilization (the default):
%usr, %sys, %wio, %idle – portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.

–**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, e.g., 1 – bread/lread;
pread/s, pwrit/s – transfers via raw (physical) device mechanism.

–**d** Report activity for each block device, e.g., disk or tape drive:
%busy, avque – portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s – number of data transfers from or to device, number of bytes transferred in 512 byte units;
avwait, avserv – average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

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

–**c** Report system calls:
scall/s – system calls of all types;
sread/s, swrit/s, fork/s, exec/s – specific system calls;
rchar/s, wchar/s – characters transferred by read and write system calls.

–**w** Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s – number of transfers and number of 512 byte units transferred for swapins (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 % 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 text, process, inode and file tables:
text-sz, proc-sz, inod-sz, file-sz – entries/size for each table, evaluated once at sampling point;
text-ov, 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 **-udqbwcayvm**.

EXAMPLE

```
sar
```

shows today's CPU activity so far.

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

watches CPU activity evolve for 10 minutes and saves data.

```
sar -d -f temp
```

later reviews disk and tape activity from that period.

FILES

/usr/adm/sa/sa *dd* daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sag(1G).
sar(1M) in the *UniPlus+ Administrator's Manual.*

NAME

sccsdiff − compare two versions of an SCCS file

SYNOPSIS

sccsdiff −r SID1 −r SID2 [−p] [−sn] files

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

−r *SID ?* *SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given.

−p pipe output for each file through *pr*(1).

−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.

EXAMPLE

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

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), get(1), help(1), pr(1).
Source Code Control System.

DIAGNOSTICS

"*file: No differences*" If the two versions are the same.
Use *help*(1) for explanations.

NAME

sdiff − side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

Sdiff uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

The following options exist:

−w *n* Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.

−l Only print the left side of any lines that are identical.

−s Do not print identical lines.

−o *output* Use the next argument, *output*, as the name of a third file that is created as a user controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed commands:

- **l** append the left column to the output file
- **r** append the right column to the output file
- **s** turn on silent mode; do not print identical lines
- **v** turn off silent mode
- **e l** call the editor with the left column
- **e r** call the editor with the right column
- **e b** call the editor with the concatenation of left and right
- **e** call the editor with a zero length file
- **q** exit from the program

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

EXAMPLE

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
```

SEE ALSO

diff(1), ed(1).

NAME

sed – stream editor

SYNOPSIS

sed [**–n**] [**–e** script] [**–f** sfile] [files]

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **–f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **–e** option and no **–f** options, the flag **–e** may be omitted. The **–n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **–n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of *ed*(1) modified thus:

In a context address, the construction **** *?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

The escape sequence **\n** matches a new-line *embedded* in the pattern space.

A period **.** matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

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

The *text* argument consists of one or more lines, all but the last of which end with **** to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before

processing begins. There can be at most 10 distinct *wfile* arguments.

(1) **a**
text Append. Place *text* on the output before reading the next input line.

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

(2) **c**
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) **d** Delete the pattern space. Start the next cycle.

(2) **D** Delete the initial segment of the pattern space through the first new-line. Start the next cycle.

(2) **g** Replace the contents of the pattern space by the contents of the hold space.

(2) **G** Append the contents of the hold space to the pattern space.

(2) **h** Replace the contents of the hold space by the contents of the pattern space.

(2) **H** Append the contents of the pattern space to the hold space.

(1) **i**
text Insert. Place *text* on the standard output.

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

(2) **n** Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) **N** Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

(2) **p** Print. Copy the pattern space to the standard output.

(2) **P** Copy the initial segment of the pattern space through the first new-line to the standard output.

(1) **q** Quit. Branch to the end of the script. Do not start a new cycle.

(2) **r** *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) **s** */ regular expression / replacement / flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed*(1). *Flags* is zero or more of:

- **g** Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
- **p** Print the pattern space if a replacement was made.
- **w** *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) **t** *label* Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.

(2) **w** *wfile* Write. Append the pattern space to *wfile*.

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

(2) **y** */ string1 / string2 /*
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1*

and *string2* must be equal.

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

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

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

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

(0) An empty command is ignored.

EXAMPLE

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

will process the "inputfile" according to the *sedfile* script, and place the results in "filea".

The *sedfile* script

```
4 a\
XXXXXXXXXXXXX
```

would insert a row of Xs after line 4.

SEE ALSO

awk(1), ed(1), grep(1).

NAME

sh, rsh − shell, the standard/restricted command programming language

SYNOPSIS

sh [−**ceiknrstuvx**] [args]
rsh [−**ceiknrstuvx**] [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.

Commands.

A *simple-command* is a sequence of non-blank *words* separated by *blanks* (a *blank* is a tab or a space). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(2) for a list of status values).

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

A *list* is a sequence of one or more pipelines separated by ;, **&**, **&&**, or |, and optionally terminated by ; or **&**. Of these four symbols, ; and **&** have equal precedence, which is lower than that of **&&** and ‖ . The symbols **&&** and ‖ also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **&&** (‖) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**
Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**
A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see *File Name Generation* below).

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list*

following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**
A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)
Execute *list* in a sub-shell.

{ *list*;}
list is simply executed.

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments.
A word beginning with **#** causes that word and all the following characters up to a new-line to be ignored.

Command Substitution.
The standard output from a command enclosed in a pair of grave accents (` `) may be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution.
The character **$** is used to introduce substitutable *parameters*. Positional parameters may be assigned values by **set**. Variables may be set by writing:

name= *value* [*name*= *value*] ...

Pattern-matching is not performed on *value*.

${*parameter*}
A *parameter* is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the characters *, **#**, ?, −, **$**, and **!**. 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. A *name* must begin with a letter or underscore. If *parameter* is a digit, then it is a positional parameter. If *parameter* is * or then 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, then substitute its value; otherwise substitute *word*.

${*parameter* := *word*}
If *parameter* is not set or is null, then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter* :? *word*}
If *parameter* is set and is non-null, then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, then the message "parameter null or not set" is printed.

${*parameter* :+ *word*}
If *parameter* is set and is non-null, then 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, then the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
-	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last synchronously executed command.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

HOME	The default argument (home directory) for the *cd* command.
PATH	The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rsh*.
CDPATH	The search path for the *cd* command.
MAIL	If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file.
PS1	Primary prompt string, by default "**$**".
PS2	Secondary prompt string, by default "**>**".
IFS	Internal field separators, normally **space**, **tab**, and **new-line**.

The shell gives default values to **PATH**, **PS1**, **PS2**, and **IFS**, while **HOME** and **MAIL** are not set at all by the shell (although **HOME** *is* set by *login*(1)).

Blank Interpretation.
After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation.
Following substitution, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears, then the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

*	Matches any string, including the null string.

? Matches any single character.

[...] Matches any one of the enclosed characters. A pair of characters separated by − matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "!", then any character not enclosed is matched.

Quoting.

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > new-line space tab

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

Prompting.

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

Input/Output.

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

< word

Use file *word* as standard input (file descriptor 0).

> word

Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.

>> word

Use file *word* as standard output. If the file exists, then 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, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\new-line** is ignored, and \ must be used to quote the characters \, **$**, `, and the first character of *word*. If − is appended to **<<**, then all leading tabs are stripped from *word* and from the document.

< & digit

The standard input is duplicated from file descriptor *digit* (see *dup*(2)). Similarly for the standard output using **>**.

< & − The standard input is closed. Similarly for the standard output using **>**.

If one of the above is preceded by a digit, then the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example:

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

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by **&**, then 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. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, 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 args                    and
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c**:

```
echo a=b c
set -k
echo a=b c
```

Signals.

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution.

Each time a command is executed, the above substitutions are carried out. Except for the *Special Commands* listed below, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the

command name contains a / then the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a sub-shell.

Special Commands.

The following commands are executed in the shell process and, except as specified, no input/output redirection is permitted for such commands:

: 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, then break *n* levels.

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

cd [*arg*]
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a /, then 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*.

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, then 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 arguments are given, then a list of all names that are exported in this shell is printed.

newgrp [*arg* ...]
Equivalent to **exec newgrp** *arg* ....

read [*name* ...]
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an

end-of-file is encountered.

readonly [*name* ...]
The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, then a list of all *readonly* names is printed.

set [−**ekntuvx** [*arg* ...]]
−**e** Exit immediately if a command exits with a non-zero exit status.
−**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, then the values of all names are printed.

shift [*n*]
The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

test Evaluate conditional expressions. See *test*(1) for usage and description.

times
Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
arg is a command 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, then all trap(s) *n* are reset to their original values. If *arg* is the null string, then this signal is ignored by the shell and by the commands it invokes. If *n* is 0, then 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.

ulimit [−**fp**] [*n*]
imposes a size limit of *n*
−**f** imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.
−**p** changes the pipe size to *n* (UNIX/RT only).
If no option is given, −**f** is assumed.

umask [*nnn*]
The user file-creation mask is set to *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

wait [*n*]
Wait for the specified process and report its termination status. If *n* is not given, then all currently active child processes are waited for and

the return code is zero.

Invocation.

If the shell is invoked through *exec*(2) and the first character of argument zero is −, commands are initially read from **/etc/profile** and then 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, then commands are read from *string*.

−s If the **−s** flag is present or if no arguments remain, then commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output 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, then this shell is *interactive*. In this case, TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

−r If the **−r** flag is present, the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

Rsh Only.

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd*(1)),
setting the value of **$PATH**,
specifying path or command names containing /,
redirecting output (> and >>).

The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively then 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).

EXAMPLE

```
sh -x script1
```

will execute each command in "script1", echoing the command just before executing it.

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cd(1), env(1), login(1), newgrp(1), test(1), umask(1), dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2), umask(2), wait(2), a.out(4), profile(4), environ(5).

BUGS

The command **readonly** (without arguments) produces the same output as the command **export**.
If << is used to provide standard input to an asynchronous process invoked by **&**, the shell gets mixed up about naming the input document; a garbage file **/tmp/sh*** is created and the shell complains about not being able to find that file by another name.

NAME

size – size of an object file

SYNOPSIS

size [**-x**] [object ...]

DESCRIPTION

Size prints the decimal number of bytes required by the text, data, and bss portions, and their sum in decimal and (hexidecimal), of each object-file argument. If no file is specified, **a.out** is used.

If the **-x** option is specified, *size* prints the hexidecimal number of bytes required by the text, data, and bss portions, and their sum in hexidecimal and (decimal).

EXAMPLE

size

prints the number of bytes for the various portions of the *a.out* file, and their sum in decimal and hexidecimal.

SEE ALSO

a.out(5).

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
```

EXAMPLE

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.

SEE ALSO

alarm(2), sleep(3C).

BUGS

Time must be less than 65536 seconds.

NAME

sno – SNOBOL interpreter

SYNOPSIS

sno [files]

DESCRIPTION

Sno is a SNOBOL compiler and interpreter (with slight differences). *Sno* obtains input from the concatenation of the named *file*s and the standard input. All input through a statement containing the label **end** is considered program and is compiled. The rest is available to **syspit**.

Sno differs from SNOBOL in the following ways:

There are no unanchored searches. To get the same effect:

```
a ** b          unanchored search for b.
a *x* b = x c   unanchored assignment
```

There is no back referencing.

```
x = "abc"
a *x* x         is an unanchored search for abc.
```

Function declaration is done at compile time by the use of the (non-unique) label **define**. Execution of a function call begins at the statement following the **define**. Functions cannot be defined at run time, and the use of the name **define** is preempted. There is no provision for automatic variables other than parameters. Examples:

```
define f( )
define f(a, b, c)
```

All labels except **define** (even **end**) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on **end** cannot merely name a label.

If **start** is a label in the program, program execution will start there. If not, execution begins with the first executable statement; **define** is not an executable statement.

There are no builtin functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators / and * must be set off by spaces.

The right side of assignments must be non-empty.

Either ' or " may be used for literal quotes.

The pseudo-variable **sysppt** is not available.

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

sort − sort and/or merge files

SYNOPSIS

sort [−**cmubdfinrtx**] [+pos1 [−pos2]] ... [−**o** output] [names]

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name − means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

b Ignore leading blanks (spaces and tabs) in field comparisons.

d "Dictionary" order: only letters, digits and blanks are significant in comparisons.

f Fold upper case letters onto lower case.

i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **n** implies option **b**.

r Reverse the sense of comparisons.

t*x* "Tab character" separating fields is *x*.

The notation +*pos1* −*pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m*.*n*, optionally followed by one or more of the flags **bdfinr**, where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first non-blank in the field; **b** is attached independently to *pos2*. A missing .*n* means .0; a missing −*pos2* means the end of the line. Under the −**t***x* option, fields are strings separated by *x*; otherwise fields are non-empty non-blank strings separated by blanks.

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.

These option arguments are also understood:

c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

m Merge only, the input files are already sorted.

u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

o The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.

EXAMPLE

```
sort −u +0f +0 list
```

prints in alphabetical order all the unique spellings in a list of words (capitalized words differ from uncapitalized).

```
sort −t: +2n /etc/passwd
```

prints the password file (*passwd*(4)) sorted by user ID (the third colon-separated field).

```
sort −um +0 −1 dates
```

print the first instance of each month in an already sorted file of (month-day) entries (the options **−um** with just one input file make the choice of a unique representative from a set of equal lines predictable).

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions and for disorder discovered under option **−c**.

BUGS

Very long lines are silently truncated.

NAME

spell, hashmake, spellin, hashcheck − find spelling errors

SYNOPSIS

spell [−**v**] [−**b**] [−**x**] [−**l**] [+local_file] [files]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

Spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

Spell ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

Under the −**v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the −**b** option, British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

Under the −**x** option, every plausible stem is printed with = for each word.

By default, *spell* (like *deroff*(1)) follows chains of included files (**.so** and **.nx** *troff*(1) requests), *unless* the names of such included files begin with **/usr/lib**. Under the −**l** option, *spell* will follow the chains of *all* included files.

Under the + *local_file* option, words found in *local_file* are removed from *spell*'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy−y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell*:

hashmake Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.

spellin Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.

hashcheck Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

EXAMPLE

spell filea fileb filec > misteaks

would put a list of the words from "filea", "fileb" and "filec" that were not part of the on-line dictionary into file "misteaks". The on-line dictionary rejects technical terms and proper names it does not know and treats them as misspellings.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/usr/lib/spell/spellhist	history file
/usr/lib/spell/spellprog	program

SEE ALSO

deroff(1), eqn(1), sed(1), sort(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 spelling feature was done by an American.

NAME

spline – interpolate smooth curve

SYNOPSIS

spline [options]

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 *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: $y_0'' = ky_1''$, $y_n'' = ky_{n-1}''$

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).

EXAMPLE

```
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
```

DIAGNOSTICS

When data is not strictly monotone in x, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1,000 input points is enforced silently.

NAME

split − split a file into pieces

SYNOPSIS

split [−*n*] [file [name]]

DESCRIPTION

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.

EXAMPLE

split -100 filea newfile

would split "filea" into 100-line pieces and put them in "newfileaa", "newfilebb", and so forth until the end of filea.

SEE ALSO

bfs(1), csplit(1).

NAME

ssp − make output single spaced

SYNOPSIS

ssp [name ...]

DESCRIPTION

Ssp removes extra blank lines and causes all output to be single spaced. It can be used directly, or as a filter after *nroff* or other text formatting operations.

EXAMPLE

nroff -ms filea fileb | ssp >> filec

would *nroff* the files with the **−ms** macro package, then single space the output and direct it to "filec".

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.

EXAMPLE

```
strings obj1
```

will locate the ASCII-character strings in the object file "obj1".

SEE ALSO

od(1).

BUGS

The algorithm for identifying strings is extremely primitive.

NAME

strip − remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and link editor. This is useful to save space after a program has been debugged.

The effect of *strip* is the same as use of the **−s** option of *ld*.

If *name* is an archive file, *strip* will remove the local symbols from any *a.out* format files it finds in the archive. Certain libraries, such as those residing in **/lib**, have no need for local symbols. By deleting them, the size of the archive is decreased and link editing performance is increased.

EXAMPLE

```
strip a.out
```

removes the symbol table and relocation bits from **a.out**.

FILES

/tmp/stm* temporary file

SEE ALSO

ld(1).

NAME

stty – set the options for a terminal

SYNOPSIS

stty [**–a**] [**–g**] [options]

DESCRIPTION

Stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options; with the **–a** option, it reports all of the option settings; with the **–g** option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first five groups below may be found in *termio*(7) for asynchronous lines, or in *stermio*(7) for synchronous lines in the *UNIX System Administrator's Manual* . Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (**–parenb**) enable (disable) parity generation and detection.

parodd (**–parodd**) select odd (even) parity.

cs5 cs6 cs7 cs8 select character size (see *termio*(7)).

0 hang up phone line immediately.

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.) **19200** is equivalent to **exta**. **38400** is equivalent to **extb**.

hupcl (**–hupcl**) hang up (do not hang up) a DATA-PHONE® data set connection on last close.

hup (**–hup**) same as **hupcl** (**–hupcl**).

cstopb (**–cstopb**) use two (one) stop bits per character.

cread (**–cread**) enable (disable) the receiver.

clocal (**–clocal**) assume a line without (with) modem control.

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) upper-case alphabetics to lower case on input.

ixon (**–ixon**) enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (**–ixany**) allow any character (only DC1) to restart output.

ixoff (**–ixoff**) request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (**−opost**) post-process output (do not post-process output; ignore all other output modes).

olcuc (**−olcuc**) map (do not map) lower-case alphabetics to upper case on output.

onlcr (**−onlcr**) map (do not map) NL to CR-NL on output.

ocrnl (**−ocrnl**) map (do not map) CR to NL on output.

onocr (**−onocr**) do not (do) output CRs at column zero.

onlret (**−onlret**) on the terminal NL performs (does not perform) the CR function.

ofill (**−ofill**) use fill characters (use timing) for delays.

ofdel (**−ofdel**) fill characters are DELs (NULs).

cr0 cr1 cr2 cr3 select style of delay for carriage returns (see *termio*(7)).

nl0 nl1 select style of delay for line-feeds (see *termio*(7)).

tab0 tab1 tab2 tab3 select style of delay for horizontal tabs (see *termio*(7) or *stermio*(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 and QUIT.

icanon (**−icanon**) enable (disable) canonical input (ERASE and KILL processing).

xcase (**−xcase**) canonical (unprocessed) upper/lower-case presentation.

echo (**−echo**) echo back (do not echo back) every character typed.

echoe (**−echoe**) echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

echok (**−echok**) echo (do not echo) NL after KILL character.

lfkc (**−lfkc**) the same as **echok** (**−echok**); obsolete.

echonl (**−echonl**) echo (do not echo) NL.

noflsh (**−noflsh**) disable (enable) flush after INTR or QUIT.

stwrap (**−stwrap**) disable (enable) truncation of lines longer than 79 characters on a synchronous line.

stflush (**−stflush**) enable (disable) flush on a synchronous line after every *write*(2).

stappl (**−stappl**) use application mode (use line mode) on a synchronous line.

Control Assignments

control-character c set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **eof**, **eol**, **ctab**, **min**, or **time** (**ctab** is used with **−stappl**; see *stermio*(7)), (**min** and **time** are used with **−icanon**; see *termio*(7)). If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., "^**d**" is a **CTRL-d**); "^**?**" is interpreted as DEL and "^**−**" is interpreted as undefined.

line *i* set line discipline to *i* ($0 < i < 127$).

Combination Modes

evenp or **parity** enable **parenb** and **cs7**.

oddp enable **parenb**, **cs7**, and **parodd**.

−parity, **−evenp**, or **−oddp**
disable **parenb**, and set **cs8**.

raw (**−raw** or **cooked**)
enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, EOT, or output post processing).

nl (**−nl**) unset (set) **icrnl**, **onlcr**. In addition **−nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

lcase (**−lcase**) set (unset) **xcase**, **iuclc**, and **olcuc**.

LCASE (**−LCASE**) same as **lcase** (**−lcase**).

tabs (**−tabs** or **tab3**)
preserve (expand to spaces) tabs when printing.

ek reset ERASE and KILL characters back to normal **#** and

sane resets all modes to some reasonable values.

term set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

SEE ALSO

tabs(1), ioctl(2).
stermio(7), termio(7) in the *UNIX System Adminstrator's Manual.*

NAME

su − become super-user or another user

SYNOPSIS

su [−] [name [arg ...]]

DESCRIPTION

Su allows one to become another user without logging off. The default user *name* is **root** (i.e., super-user).

To use *su*, the appropriate password must be supplied (unless one is already super-user). If the password is correct, *su* will execute a new shell with the user ID set to that of the specified user. To restore normal user ID privileges, type an **EOF** to the new shell.

Any additional arguments are passed to the shell, permitting the super-user to run shell procedures with restricted privileges (an *arg* of the form **−c** *string* executes *string* via the shell). When additional arguments are passed, **/bin/sh** is always used. When no additional arguments are passed, *su* uses the shell specified in the password file.

An initial − flag causes the environment to be changed to the one that would be expected if the user actually logged in again. This is done by invoking the shell with an *arg0* of **−su** causing the **.profile** in the home directory of the new user ID to be executed. Otherwise, the environment is passed along with the possible exception of **$PATH**, which is set to **/bin:/etc:/usr/bin** for root. Note that the **.profile** can check *arg0* for **−sh** or **−su** to determine how it was invoked.

EXAMPLE

```
su unisoft
```

would cause the system to prompt for UniSoft's password; if the password is typed in correctly, UniSoft's identity is substituted for yours, so far as the system is concerned.

FILES

/etc/passwd	system's password file
$HOME/.profile	user's profile

SEE ALSO

env(1), login(1), sh(1), environ(5).

NAME

sum − print checksum and block count of a file

SYNOPSIS

sum [**−r**] file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **−r** causes an alternate algorithm to be used in computing the checksum.

EXAMPLE

```
sum filea
```

produces the checksum and the block count of "filea".

SEE ALSO

wc(1).

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NAME

sum7 – sum and count blocks in a file

SYNOPSIS

sum7 file

DESCRIPTION

Sum7 calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file, to the nearest whole block. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

EXAMPLE

```
sum7 sum7.1
```

produces the checksum and the block count of this manual section, namely:

```
21009   1
```

SEE ALSO

wc(1).

NAME

sumdir − sum and count characters in the files in the given directories

SYNOPSIS

sumdir [directories]

DESCRIPTION

Sumdir 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.

Sumdir provides a recursive checksum of all files in the specified directory.

EXAMPLE

```
sumdir man1
```

produces the checksum and the character count of the files in the directory **man1**.

SEE ALSO

sum(1).

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync*(2) for details.

EXAMPLE

```
sync
```

should be typed to flush all internal disk buffers, before bringing down the system.

SEE ALSO

sync(2).

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 of course be logged in on a terminal with remotely-settable hardware tabs.

Users of GE TermiNet terminals should be aware that they behave in a different way than most other terminals for some tab settings: the first number in a list of tab settings becomes the *left margin* on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

Four types of tab specification are accepted for *tabspec*: "canned", repetitive, arbitrary, and file. If no *tabspec* is given, the default value is **−8**, i.e., UNIX "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.

−*code* Gives the name of one of a set of "canned" tabs. The legal codes and their meanings are as follows:

−a 1,10,16,36,72
Assembler, IBM S/370, first format

−a2 1,10,16,40,72
Assembler, IBM S/370, second format

−c 1,8,12,16,20,55
COBOL, normal format

−c2 1,6,10,14,49
COBOL, compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:

<:t−c2 m6 s66 d:>

−c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than **−c2**. This is the recommended format for COBOL. The appropriate format specification is:

<:t−c3 m6 s66 d:>

−f 1,7,11,15,19,23
FORTRAN

−p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

−s 1,10,55
SNOBOL

−u 1,12,20,44
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

$-n$ A repetitive specification requests tabs at columns $1+n$, $1+2*n$, etc. Note that such a setting leaves a left margin of n columns on TermiNet terminals *only*. Of particular importance is the value **−8**: this represents the UNIX "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the *nroff* **−h** option for high-speed output. Another special case is the value **−0**, implying no tabs 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.

EXAMPLE

tabs −c

will send commands to the terminal to remotely set the tabs for COBOL format.

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 "canned" code cannot be found.
can't open	if − −*file* option used, and file can't be opened.
file indirection	if − −*file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted.

SEE ALSO

nroff(1), environ(4), term(5), tset(1).

BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

It is generally impossible to usefully change the left margin without also setting tabs.

Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 40.

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") 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.

EXAMPLE

```
tail −f fred
```

will print the last ten lines of the file "fred", followed by any lines that are appended to "fred" between the time *tail* is initiated and killed.

```
tail −15cf fred
```

will print the last 15 characters of the file "fred", followed by any lines that are appended to "fred" between the time *tail* is initiated and killed.

SEE ALSO

dd(1).

BUGS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

NAME

take − takes a file from a remote machine

SYNOPSIS

take [**−p** *port*] [−sSPEED] [**−i** [ID]] *fromfile* [*tofile*]
take [**−p** *port*] [−sSPEED] **−c** *command* [*args*] ...

DESCRIPTION

Take is part of system of programs useful for transferring files between UNIX systems. It is the "downloader" designed to transmit files from a remote machine to a local machine. For a brief discussion of the take/put system and installation instructions, see the companion document: *Installation and Overview of the UniSoft Take/Put File Transfer System.*

Take transfers a file, directory, or output from a command given at a remote machine. The default *port* is **/dev/tty0**; the **−p** option can be used to specify an alternate port. The default *speed* is determined by the system; the **−s** option can be used to specify a specific speed. The **−i** [ID] option remaps pathnames on the remote machine. The **ID** (if present) is passed to the remote machine where it is used to locate a line containing pathname prefixes (using the **/etc/takelist** file discussed below). If no **ID** is given after the **−i** flag, then the default system ID is read from the **/etc/sys_id** file (if it exists); otherwise **take7** will use the account name of the invoker (i.e., the person who logged in to the port used) to determine which line of **/etc/takelist** to apply. See the overview document for details of the mapping.

The **−c** option is useful for executing an arbitrary command on the remote machine. All arguments following the **−c** flag are collected, transmitted to the remote machine and executed as a single command. The standard output and standard error from this command are returned as the standard output and standard error of **take**.

In order to perform its function, *take*(1C) interfaces with the program **/usr/bin/take7** on the remote machine.

EXAMPLE

```
take /a/b/c
```

takes the contents of the directory (or file) "/a/b/c" on the remote machine and copies them into a similarly named directory (or file) on the local machine; if "/a/b/c" did not previously exist on the local machine, it is created; otherwise it is overwritten.

```
take file.c /x/y/z/filename
```

takes the contents of "file.c" from the remote machine and copies them into "/x/y/z/filename" on the local machine. Note that if "filename" exists on the local machine, its contents are overwritten.

FILES

fromfile — The remote file name. When using the **−i** option, this file should usually be specified as a pathname starting at the root of the local machine.

tofile — The local file name; if *tofile* is null, *tofile* is defaulted to *fromfile*. If *tofile* is a directory, then *tofile* has the last segment of the *fromfile* path appended to it.

SEE ALSO

cu(1C), put(1C)
Installation and Overview of the UniSoft Take/Put File Transfer System

NAME

take7 − takes a file from a remote machine.

SYNOPSIS

take7 [**−p** *port*] [**−sSPEED**] [**−i[ID]**] *fromfile* [*tofile*]
take7 [**−p** *port*] [**−sSPEED**] **−c** *command* [*args*] ...

DESCRIPTION

Take7 is part of system of programs useful for transferring files between UNIX systems. It is the "downloader" designed to transmit files from a remote machine to a local machine. For a brief discussion of the take/put system and installation instructions, see the companion document: *Overview of the UniSoft Take/Put File Transfer System.*

Take7 transfers a file, directory, or output from a command given at a remote machine. The default port is **/dev/tty0**; the **−p** option can be used to specify an alternate port. The default *speed* is determined by the system; the **−s** option can be used to specify a specific speed. The **−i[ID]** option remaps pathnames on the remote machine. The **ID** (if present) is passed to the remote machine where it is used to locate a line containing pathname prefixes (using the **/etc/takelist** file discussed below). If no **ID** is given after the **−i** flag, then the default system ID is read from the **/etc/sys_id** file (if it exists); otherwise **take6** will use the account name of the invoker (i.e., the person who logged in to the port used) to determine which line of **/etc/takelist** to apply. See the overview document for details of the mapping.

The **−c** option is useful for executing an arbitrary command on the remote machine. All arguments following the **−c** flag are collected, transmitted to the remote machine and executed as a single command. The standard output and standard error from this command are returned as the standard output and standard error of **take7**.

In order to perform its function, *take7*(1C) interfaces with the program **/usr/bin/take6** on the remote machine.

EXAMPLE

```
take7 /a/b/c
```

takes the contents of the directory (or file) "/a/b/c" on the remote machine and copies them into a similarly named directory (or file) on the local machine; if "/a/b/c" did not previously exist on the local machine, it is created; otherwise it is overwritten.

```
take file.c /x/y/z/filename
```

takes the contents of "file.c" from the remote machine and copies them into "/x/y/z/filename" on the local machine. Note that if "filename" exists on the local machine, its contents are overwritten.

FILES

fromfile — The remote file name. When using the **−i** option, this file should usually be specified as a pathname starting at the root of the local machine.

tofile — The local file name; if *tofile* is null, *tofile* is defaulted to *fromfile*. If *tofile* is a directory, then *tofile* has the last segment of the *fromfile* path appended to it.

SEE ALSO

cu(1), put7(1)

Overview of the UniSoft Take/Put File Transfer System

NAME

tar − tape file archiver

SYNOPSIS

tar [key] [files]

DESCRIPTION

Tar saves and restores files on magnetic tape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named *files* are written on the end of the tape. The **c** function implies this function.

x The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode 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 the specified files are listed each time that they occur on the tape. If no *files* argument is given, all the names 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:

0,...,7 This modifier selects the drive on which the tape is mounted. The default is **1**.

v Normally, *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w causes *tar* to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with **y** is given, the action is performed. Any other input means "no".

f causes *tar* to use the next argument as the name of the archive instead of **/dev/mt?**. If the name of the file is −, *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *Tar* can also be used to move hierarchies with the command:

```
cd fromdir; tar cf − . | (cd todir; tar xf −)
```

b causes *tar* to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes (key letters **x** and **t**).

l tells *tar* to complain if it cannot resolve all of the links to the files being dumped. If **l** is not specified, no error messages are printed.

m tells *tar* to not restore the modification times. The modification time of the file will be the time of extraction.

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 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 1600BPI.

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 in the disk.

The *tar* header format is as follows:

```
# define TBLOCK  512
# define NBLOCK  40
# define NAMSIZ  100
union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMESIZ];
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char linkflag;
                char linkname[NAMESIZ];
        } dbuf;
} dblock, tbuf[NBLOCK];
```

EXAMPLE

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

will copy directories from one directory tree to another.

FILES

/dev/rmt?
/dev/mt?
/tmp/tar*
/bin/mkdir build directories during recovery
/bin/pwd get working directory name

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

BUGS

There is no way to ask for the *n*-th occurrence of a file.
Tape errors are handled ungracefully.
The **u** option can be slow.
The **b** option should not be used with archives that are going to be updated.
The current magnetic tape driver cannot backspace raw magnetic tape. If

the archive is on a disk file, the **b** option should not be used at all, because updating an archive stored on disk can destroy it.
The current limit on file-name length is 100 characters.
Empty directories are skipped when creating a tar archive.

NAME

tbl − format tables for nroff or troff

SYNOPSIS

tbl [−TX] [files]

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 re-formatted 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 semi-colon (;).

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 key-letter, 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, inter-column spacing, etc. The available key-letters 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** option forces *tbl* to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (e.g., 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.

EXAMPLE

In the following input, ^I represents a tab (which should be typed as a genuine tab):

```
.TS
center box ;
cB s s
cI | cI s
^ | c c
l | n n .
Household Population

Town^IHouseholds
^INumber^ISize
=
Bedminster^I789^I3.26
Bernards Twp.^I3087^I3.74
Bernardsville^I2018^I3.30
Bound Brook^I3425^I3.04
Bridgewater^I7897^I3.81
Far Hills^I240^I3.19
.TE
```

yields:

Household Population		
Town	*Households*	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

SEE ALSO

cw(1), eqn(1), mm(1), mmt(1), nroff(1), troff(1), mm(5), mv(5)
TBL—A Program to Format Tables.

BUGS

See *BUGS* under *nroff*(1).

NAME

tc − phototypesetter simulator

SYNOPSIS

tc [**−t**] [**−sn**] [**−pl**] [file]

DESCRIPTION

Tc interprets its input (standard input default) as device codes for a Wang Laboratories, Inc. C/A/T phototypesetter. The standard output of *tc* is intended for a Tektronix 4014 terminal with ASCII and APL character sets. The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, with overstruck combinations where necessary. Typical usage is:

troff −t files | tc

At the end of each page, *tc* waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command **e** will *suppress* the screen erase before the next page; **s** *n* will cause the next *n* pages to be skipped; and **!***cmd* will send *cmd* to the shell.

The command line options are:

−t Don't wait between pages (for directing output into a file).

−s*n* Skip the first *n* pages.

−p*l* Set page length to *l*; *l* may include the scale factors **p** (points), **i** (inches), **c** (centimeters), and **P** (picas); default is picas.

SEE ALSO

4014(1), sh(1), tplot(1G), troff(1).

BUGS

Font distinctions are lost.

NAME

tee − pipe fitting

SYNOPSIS

tee [**−i**] [**−a**] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*. The **−i** option ignores interrupts; the **−a** option causes the output to be appended to the *files* rather than overwriting them.

EXAMPLE

```
make | tee x
```

will cause the output of the make program to be recorded on file "x" as well as printed on standard output.

NAME

test − condition evaluation command

SYNOPSIS

test expr
[expr]

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

−r *file*	true if *file* exists and is readable.
−w *file*	true if *file* exists and is writable.
−x *file*	true if *file* exists and is executable.
−f *file*	true if *file* exists and is a regular file.
−d *file*	true if *file* exists and is a directory.
−c *file*	true if *file* exists and is a character special file.
−b *file*	true if *file* exists and is a block special file.
−p *file*	true if *file* exists and is a named pipe (fifo).
−u *file*	true if *file* exists and its set-user-ID bit is set.
−g *file*	true if *file* exists and its set-group-ID bit is set.
−k *file*	true if *file* exists and its sticky bit is set.
−s *file*	true if *file* exists and has a size greater than zero.
−t [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
−z *s1*	true if the length of string *s1* is zero.
−n *s1*	true if the length of the string *s1* is non-zero.
s1 = *s2*	true if strings *s1* and *s2* are identical.
s1 != *s2*	true if strings *s1* and *s2* are *not* identical.
s1	true if *s1* is *not* the null string.
n1 **−eq** *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **−ne**, **−gt**, **−ge**, **−lt**, and **−le** may be used in place of **−eq**.

These primaries may be combined with the following operators:

!	unary negation operator.
−a	binary *and* operator.
−o	binary *or* operator (**−a** has higher precedence than **−o**).
(expr)	parentheses for grouping.

Notice 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.

EXAMPLE

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*ed.

```
if (test -d foo) then
        echo "foo is a dir"
fi
```

SEE ALSO

find(1), sh(1).

WARNING

In the second form of the command (i.e., the one that uses [], rather than the word *test*), the square brackets must be delimited by blanks.
Some UNIX systems do not recognize the second form of the command.

Note *test* is built into **/bin/sh** and will not work in **/bin/csh**.

NAME

time – time a command

SYNOPSIS

time command

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The execution time can depend on what kind of memory the program happens to land in; the user time in MOS is often half what it is in core.

The times are printed on standard error.

EXAMPLE

```
time nroff man 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*, on the other hand, the time report might be:

```
8.9u 7.0s 0:29 54%
```

which reports 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 real elapsed time.

SEE ALSO

timex(1), times(2).

NAME

timex − time a command; report process data and system activity

SYNOPSIS

timex [options] command

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of *timex* is written on standard error.

Options are:

−p List process accounting records for *command* and all its children. Suboptions **f**, **h**, **k**, **m**, **r**, and **t** modify the data items reported, as defined in *acctcom*(1). The number of blocks read or written and the number of characters transferred are always reported.

−o Report the total number of blocks read or written and total characters transferred by *command* and all its children.

−s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

EXAMPLE

timex ps -el

runs the *ps* command (with the correct options), then produces statistics concerning the command and system activity during the command to the standard error.

SEE ALSO

acctcom(1), sar(1).

WARNING

Process records associated with *command* are selected from the accounting file **/usr/adm/pacct** by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

NAME

touch − update access and modification times of a file

SYNOPSIS

touch [**−amc**] [mmddhhmm[yy]] files

DESCRIPTION

Touch causes the access and modification times of each argument to be updated. If no time is specified (see *date*(1)) the current time is used. The **−a** and **−m** options cause touch to update only the access or modification times respectively (default is **−am**). The **−c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

Note that 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.

EXAMPLE

```
touch filea fileb
```

sets the "date last modified" of the two files to the current date.

SEE ALSO

date(1), utime(2).

NAME

tp − manipulate tape archive

SYNOPSIS

tp [key] [name ...]

DESCRIPTION

Tp saves and restores files on DECtape or other magnetic tape. Its actions are controlled by the *key* argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are 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) 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 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 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.

EXAMPLE

```
tp x file1
```

extracts "file1" from a *tp* formatted magnetic tape mounted on drive 0.

FILES

/dev/tap?
/dev/mt?

SEE ALSO

ar(1), cpio(1), tar(1).

DIAGNOSTICS

Several; the non-obvious 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 − graphics filters

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 *terminal*s are:

300 DASI 300.
300S DASI 300s.
450 DASI 450.
4014 Tektronix 4014.
ver Versatec D1200A. This version of *plot* places a scan-converted image in **/usr/tmp/raster$$** and sends the result directly to the plotter device, rather than to the standard output. The −**e** option causes a previously scan-converted file *raster* to be sent to the plotter.

EXAMPLE

```
tplot -T4014 graph.out
```

will use the encoded information in "graph.out" to plot a graph on a Tektronix 4014-type terminal.

FILES

/usr/lib/t300
/usr/lib/t300s
/usr/lib/t450
/usr/lib/t4014
/usr/lib/vplot
/usr/tmp/raster$$

SEE ALSO

plot(3X), plot(4), term(5).

NAME

tr − translate characters

SYNOPSIS

tr [**−cds**] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **−cds** may be used:

−c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.

−d Deletes all input characters in *string1*.

−s Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a−z] Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.

[a* *n*] Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

EXAMPLE

```
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.

In this case, *tr* has substituted the *newline* character for all the alphabetics in "file1", reconstituted the alphabetics with the *−c* option, squeezed the newlines to one per occurrence, with the *−s* option, and directed the output to "file2".

SEE ALSO

ed(1), sh(1), ascii(5).

BUGS

Won't handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

NAME

troff − typeset text

SYNOPSIS

troff [options] [files]

DESCRIPTION

Troff formats text contained in *files* (standard input by default) for a Wang Laboratories, Inc., C/A/T phototypesetter. Its capabilities are described in the *NROFF/TROFF User's Manual* cited below.

An argument consisting of a minus (−) is taken to be a file name corresponding to the standard input. The *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. *Troff* will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed.
- **−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 non-compacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.
- **−c***name* Prepend to the input *files* the compacted macro files **/usr/lib/macros/cmp.[nt].[dt].***name* and **/usr/lib/macros/ucmp.[nt].***name*.
- **−k***name* Compact the macros used in this invocation of *troff*, placing the output in files **[dt].***name* in the current directory (see the May 1979 Addendum to the *NROFF/TROFF User's Manual* for details of compacting macro files).
- **−t** Direct output to the standard output instead of the phototypesetter.
- **−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.
- **−a** Send a printable ASCII approximation of the results to the standard output.
- **−p***N* Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.
- **−T***name* Use font-width tables for device *name* (the font tables are found in **/usr/lib/font/***name*/*). Currently, no *name*s are supported.

EXAMPLE

```
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

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/font/*	font width tables for *troff*

SEE ALSO

cw(1), eqn(1), mmt(1), nroff(1), tbl(1), tc(1), mm(5), mv(5).
NROFF/TROFF User's Manual and *A TROFF Tutorial*

BUGS

Troff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *troff* generates may be off by one day from your idea of what the date is.
When *troff* is used with the **−o***list* option inside a pipeline (e.g., 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

troff7 − text formatting and typesetting

SYNOPSIS

troff7 [option] ... [file] ...

DESCRIPTION

Troff7 formats text in the named *files* for printing on a Graphic Systems C/A/T phototypesetter; *nroff* is used for for typewriter-like devices. Their capabilities are described in the *Nroff/Troff User's Manual.*

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (−) is taken to be a file name corresponding to the standard input. The 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.

−n*N* Number first generated page *N*.

−s*N* Stop every *N* pages. *Nroff* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline. *Troff7* will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed.

−m*name* Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

−r*aN* Set register *a* (one-character) to *N*.

−i Read standard input after the input files are exhausted.

−q Invoke the simultaneous input-output mode of the **rd** request.

Troff7 only

−t Direct output to the standard output instead of the phototypesetter.

−f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

−w Wait until phototypesetter is available, if currently busy.

−b Report whether the phototypesetter is busy or available. No text processing is done.

−a Send a printable ASCII approximation of the results to the standard output.

−pN Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.

−g Prepare output for a GCOS phototypesetter and direct it to the standard output (see *gcat*(1)).

If the file */usr/adm/tracct* is writable, *troff7* keeps phototypesetter accounting records there. The integrity of that file may be secured by making *troff7* a "set-user-id" program.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff7*
/usr/lib/font/*	font width tables for *troff7*
/dev/cat	phototypesetter
/usr/adm/tracct	accounting statistics for **/dev/cat**

SEE ALSO

eqn(1), tbl(1)
Nroff/Troff User's Manual by J. F. Ossanna,
A TROFF Tutorial by B. W. Kernighan.

NAME

true, false — provide truth values

SYNOPSIS

true

false

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1).

EXAMPLE

```
while true
do
        command
done
```

SEE ALSO

sh(1).

DIAGNOSTICS

True has exit status zero, *false* nonzero.

NAME

tset, reset − set or reset the teletype bits to a sensible state

SYNOPSIS

tset [options]
reset

DESCRIPTION

Tset causes terminal dependent processing such as setting erase and kill characters, setting or resetting delays, and the like. 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 re-initializes the terminal. The rest of this manual concerns itself with type initialization, done typically once at login, and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script ".profile" (for *sh*(1) users) or ".login" (for *csh*(1) users), it is desirable to give information about the types of terminal usually used, for terminals which are connected to the computer through a modem. These ports are initially identified as being *dialup* or *plugboard* or *arpanet* etc. To specify what terminal type is usually used on these ports, **−m** is followed by the appropriate port type identifier, an optional baud-rate 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.

Baud rates 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 rate 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** 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 (otherwise) a dialup (i.e., at 300 baud or less). If the *type* above begins with a question mark, the user is asked if s/he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. Thus, in this case, the user will be queried on a plugboard port as to whether they are using an *adm3a*. For other ports the port type will be taken from the **/etc/ttytype** file or a final, default *type* 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** options, and information about the terminal to a shell's environment. This can be done using the **−s** 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 the terminal type is known, *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.

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 options are:

−e set the erase character to be the named character *c* on all terminals, the default being the backspace character on the terminal, usually ^H.

−k is similar to **−e** but for the line kill character rather than the erase character; *c* defaults to ^X (for purely historical reasons); ^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.

Reset sets the terminal to cooked mode, turns off "cbreak" and "raw" modes, turns on "nl", and restores special characters that are undefined to their default values.

This is most useful after a program dies leaving a terminal in a funny state; you have to type **<LF>reset<LF>** to get it to work as <CR> often doesn't work; often none of this will echo.

It isn't a bad idea to follow *reset* with *tset*.

EXAMPLE

A typical *csh* ".login" file using *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

/etc/ttytype terminal id to type map database
/etc/termcap terminal capability database

SEE ALSO

csh(1), sh(1), stty(1), environ(4), ttytype(4), termcap(5).

BUGS

Should be merged with *stty*(1).

Reset doesn't set tabs properly; it can't intuit personal choices for interrupt and line kill characters, so it leaves these the old UNIX standards ^? (delete) for interrupt and @ for line kill.

It could well be 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.

NOTES

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.

AUTHOR

Eric Allman

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.

EXAMPLE

```
ar cr library `lorder *.o | tsort`
```

intends to build a new library from existing **.o** files.

SEE ALSO

lorder(1).

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty − get the terminal's name

SYNOPSIS

tty [−l] [−s]

DESCRIPTION

Tty prints the path name of the user's terminal. The **−l** option prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line. The **−s** option inhibits printing of the terminal's path name, allowing one to test just the exit code.

EXAMPLE

```
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.

NAME

ul − do underlining

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, **/etc/termcap** 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.

EXAMPLE

```
ul file1
```

displays "file1" on the terminal with underlined portions of the file either underlined, or in reverse video when this option is supported for the terminal.

FILES

/bin/cat	concatenate and print
/etc/termcap	terminal capability data base

SEE ALSO

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.

AUTHOR

Mark Horton

NAME

umask – set file-creation mode mask

SYNOPSIS

umask [000]

DESCRIPTION

The user file-creation mode mask is set to *000*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode 777 become mode **755**; files created with mode **666** become mode **644**).

If *000* is omitted, the current value of the mask is printed.

Umask is recognized and executed by the shell.

EXAMPLE

```
umask 22
```

sets file-creation mode mask such that at file creation, the *write* bits will be zeroed out for *group* and *other* users, regardless of mode specification in *create*.

SEE ALSO

chmod(1), sh(1), chmod(2), creat(2), umask(2).

NAME

uname − print name of current UNIX System

SYNOPSIS

uname [**−snrvma**]

DESCRIPTION

Uname prints the current system name of the UNIX System on the standard output file. It is mainly useful to determine what system one is using. The options cause selected information returned by *uname*(2) to be printed:

−s print the system name (default).

−n print the nodename (the nodename may be a name that the system is known by to a communications network).

−r print the operating system release.

−v print the operating system version.

−m print the machine hardware name.

−a print all the above information.

Arguments not recognized default the command to the **−s** option.

EXAMPLE

```
uname
```

on UniPlus+ would print on the screen

```
unix
```

SEE ALSO

uname(2).

NAME

unget − undo a previous get of an SCCS file

SYNOPSIS

unget [−r SID] [−s] [−n] files

DESCRIPTION

Unget undoes the effect of a **get** −e done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of − is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

−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 *get*s 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.

EXAMPLE

```
% unget s.test1.c
1.2
```

undoes version 1.2 of "test1.c" set up for editing by an earlier **get**-e.

SEE ALSO

delta(1), get(1), sact(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

uniq − report repeated lines in a file

SYNOPSIS

uniq [−**udc** [+n] [−n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the −**u** flag is used, just the lines that are not repeated in the original file are output. The −**d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the −**u** and −**d** mode outputs.

The −**c** option supersedes −**u** and −**d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

−*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

EXAMPLE

```
uniq file1
```

prints contents of "file1" with adjacent identical lines removed.

SEE ALSO

comm(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, as in the examples below.

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 below).

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, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

EXAMPLE

```
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/lib/unittab

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/rXXXXX**.

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

EXAMPLE

```
updater d . .
```

uses **/dev/tty0** to communicate with a remote machine and compares directories on the remote and local systems.

NAME

uucp, uulog, uuname — unix to unix copy

SYNOPSIS

uucp [options] source-files destination-file

uulog [options]

uuname [**-l**]

DESCRIPTION

Uucp.

Uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names which *uucp* knows about. The *system-name* may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file via the specified route, and only to a destination in PUBDIR (see below). Care should be taken to insure that intermediate nodes in the route are willing to forward information.

The shell metacharacters **?**, ***** and **[...]** appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

(1) a full path name;

(2) a path name preceded by *~user* where *user* is a login name on the specified system and is replaced by that user's login directory;

(3) a path name preceded by *~/user* where *user* is a login name on the specified system and is replaced by that user's directory under PUBDIR;

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system, the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

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 options are interpreted by *uucp*:

-d Make all necessary directories for the file copy (default).

-f Do not make intermediate directories for the file copy.

-c Use the source file when copying out rather than copying the file to the spool directory (default).

-C Copy the source file to the spool directory.

-m *file* Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

-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 only be successful if the remote machine allows the *uucp* command to be executed by **/usr/lib/uucp/uuxqt**.)

Uucp returns on the standard output a string which is the job number of the request. This job number can be used by *uustat* to obtain status or terminate the job.

Uulog.

Uulog queries a summary log of *uucp* and *uux*(1C) transactions in the file **/usr/spool/uucp/LOGFILE**.

The options cause *uulog* to print logging information:

-s *sys* Print information about work involving system *sys*.

-u *user* Print information about work done for the specified *user*.

Uuname.

Uuname lists the uucp names of known systems. The **-l** option returns the local system name.

EXAMPLE

```
uucp file1 unisoft! /usr/spool/uucppublic/file2
```

sends "file1" from the local machine, via the uucp network, to the "unisoft" machine, where it is saved as file "/usr/spool/uucppublic/file2".

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory for receiving and sending (PUBDIR)
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail(1), uux(1C).

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to ~**nuucp** or just ~).

BUGS

All files received by *uucp* will be owned by *uucp*.

The **-m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters ? * [...] will not activate the **-m** option.

NAME

uustat − uucp status inquiry and job control

SYNOPSIS

uustat [options]

DESCRIPTION

Uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. The following *options* are recognized:

−j *jobn* Report the status of the *uucp* request *jobn*. If **all** is used for *jobn*, the status of all *uucp* requests is reported. If *jobn* is omitted, the status of the current user's *uucp* requests is reported.

−k *jobn* Kill the *uucp* request whose job number is *jobn*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

−r *jobn* Rejuvenate *jobn*. That is *jobn* is touched so that its modification time is set to the current time. This prevents *uuclean* from deleting the job until the jobs modification time reaches the limit imposed by *uuclean*.

−c *hour* Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the super-user.

−u *user* Report the status of all *uucp* requests issued by *user*.

−s *sys* Report the status of all *uucp* requests which communicate with remote system *sys*.

−o *hour* Report the status of all *uucp* requests which are older than *hour* hours.

−y *hour* Report the status of all *uucp* requests which are younger than *hour* hours.

−m *mch* Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local *uucp* are provided.

−M *mch* This is the same as the *−m* option except that two times are printed. The time that the last status was obtained and the time that the last successful transfer to that system occurred.

−O Report the *uucp* status using the octal status codes listed below. If this option is not specified, the verbose description is printed with each *uucp* request.

−q List the number of jobs and other control files queued for each machine and the time of the oldest and youngest file queued for each machine. If a lock file exists for that system, its date of creation is listed.

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user. Note that only one of the options **−j**, **−m**, **−k**, **−c**, **−r**, can be used with the rest of the other options.

For example, the command:

uustat −uhdc −smhtsa −y72

will print the status of all *uucp* requests that were issued by user *hdc* to communicate with system *mhtsa* within the last 72 hours. The meanings of the job request status are:

job-number user remote-system command-time status-time status

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

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/spool/uucp	spool directory
/usr/lib/uucp/L_stat	system status file
/usr/lib/uucp/R_stat	request status file

SEE ALSO

uucp(1C).

NAME

uuto, uupick − public UNIX System-to-UNIX System file copy

SYNOPSIS

uuto [options] source-files destination
uupick [−**s** system]

DESCRIPTION

Uuto sends *source-files* to *destination*. *Uuto* uses the *uucp*(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!*user*

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*). *Logname* is the login name of someone on the specified system.

Two *options* are available:

−**p** Copy the source file into the spool directory before transmission.
−**m** Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by *mail*(1) of the arrival of files.

Uupick accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from *system*: [file *file-name*] [dir *dirname*] ?

Uupick then reads a line from the standard input to determine the disposition of the file:

<new-line>	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir* (current directory is default).
a [*dir*]	Same as **m** except moving all the files sent from *system*.
p	Print the content of the file.
q	Stop.
EOT (control-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

Uupick invoked with the −s*system* option will only search the PUBDIR for files sent from *system*.

EXAMPLE

uuto -p file1 file2 file3 ucbvax!Joe

would send the three files to user Joe on ucbvax

uupick [executed by Joe]

would tell him what has arrived and from where.

FILES

PUBDIR/usr/spool/uucppublic public directory

SEE ALSO

mail(1), uucp(1C), uustat(1C), uux(1C)
uuclean(1M) in the *UniPlus+ Administrator's Manual.*

NAME

uux − unix to unix command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

Uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail (see *mail*(1)) via *uux*.

The *command-string* is made up of one or more arguments that look like a Shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of

(1) a full path name;

(2) a path name preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

Uux will attempt to get all files to the execution system. For files which are output files, the file name must be escaped using parentheses.

Uux will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine.

The following *options* are interpreted by *uux*:

− The standard input to *uux* is made the standard input to the *command-string*.

−n Send no notification to user.

−m*file* Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

Uux returns an ASCII string on the standard output which is the job number. This job number can be used by *uustat* to obtain the status or terminate a job.

EXAMPLE

```
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/lib/uucp/L.sys List of system names and when to call them

/usr/lib/uucp/L-cmd	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

SEE ALSO

uucp(1C)
uuclean(1M) in the *UniPlus+ Administrator's Manual.*

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.

NAME

val − validate SCCS file

SYNOPSIS

val −
val files

DESCRIPTION

Val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a −, and named files.

Val has a special argument, −, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

−s The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

−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 = can't 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.

EXAMPLE

```
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.

SEE ALSO

admin(1), delta(1), get(1), prs(1).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Val can process up to 50 files on a single command line. Any number above 50 will produce a **core** dump.

NAME

vc – version control

SYNOPSIS

vc [–**a**] [–**t**] [–**c***char*] [–**s**] [keyword=value ... keyword=value]

DESCRIPTION

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the –**t** keyletter (see below). The default control character is colon (:), except as modified by the –**c** keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less 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 *tab* 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.

EXAMPLE

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=UniSoft < note
```

will produce

```
Joe,
Just a note to remind you that we have a meeting
scheduled Monday morning at UniSoft.
```

DIAGNOSTICS

Use *help*(1) for explanations.

EXIT CODES

0 – normal
1 – any error

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: "pre history". *Version* is useful for determining which version of the current program you are running.

EXAMPLE

version /bin/version

prints the version number of the version program.

NAME

vi, view − screen oriented (visual) display editor based on ex

SYNOPSIS

vi [−**t** tag] [−**r**] [+*command*] [−**w***n*] name ...
view [−**t** tag] [−**r**] [+*command*] [−**w***n*] name ...

DESCRIPTION

Vi (visual) is a display oriented text editor based on *ex*(1). *Ex* and *vi* run the same code; it is possible to get to the command mode of *ex* from within *vi* and vice-versa.

Vi puts up a screenful of text at a time (unless a smaller window is specified) and allows rapid and fluid cursor motion to the place where you want to begin adding, changing, or deleting text. With *vi*, editing can be done on characters, words, lines, or sections at a time. When multi-character changes are made, it is necessary to hit the ESCAPE key to return to cursor motion mode.

View is an invocation of *vi* which disallows writing. *View* is useful for browsing through a file when no modifications are intended.

Using *ex* commands and calling up the Shell by typing (!) are done with a colon (:) and the appropriate command sequence, such as that to find a string or write the file.

The *Vi Command Summary* (below), the *Vi Quick Reference* card and the *Introduction to Display Editing with Vi* provide full details on using *vi*.

The following options are recognized:

−**t** Equivalent to an initial *tag* command, editing the file containing the *tag* and positioning the editor at its definition.

−**r** Used in recovering after an editor or system crash, retrieving the last saved version of the named file. If no file is specified, a list of saved files will be reported.

+*command* indicates that the editor should begin by executing the specified command. If *command* is omitted, then it defaults to $, positioning the editor at the last line of the first file initially. Other useful commands here are scanning patterns of the form "/pat" or line numbers, e.g., "+100" to start at line 100.

−**w***n* sets the default window size to *n*, and is useful in dialups, to start in small windows.

Name arguments indicate files to be edited.

Vi Command Summary

Cursor Motion:	Forward	Back
letter	(space)	^H, h
word right-limit	E,e	
word left-limit	W,w	B,b
sentence	)	(
paragraph	}	{
section/function	]]	[[
line: same/limit	$	0
1st charac	+,<ret>	-
same column	^n,LF	^p

specified	<line#>G	<line#>G
1/2 screenful	^d	^u
screenful	^f	^b

Undoing Errors (see also: change, insert, delete)

u	undo last change
U	restore current line
"Np	retrieve Nth last delete
<esc>	abandon incomplete command (without completing it)
:q!	drastic! abandon without saving.

Insert		**Change**	
i	before cursor	cw<newword>	change word to newword
I	before 1st non-blank	C	change rest of line
a	after cursor	s	substitute character
A	at end-of-line	S	substitute lines
o	open line below	rx	replace 1 character
O	open line above	R	replace characters
<esc>	terminates insert	xp	transpose character
		<esc>	terminates change

Delete		**Delete during Insert**	
x	character	last character	^H
X	...before cursor	last word	^W
dw	word	all input this line	@
de	...but leave punctuation		
dd	line		
(#)dd	number of lines		
D	rest of line		

FILES

See *ex*(1).

EXAMPLE

vi text

would invoke the editor with the file named "text". For further examples, see *An Introduction to Display Editing with Vi.*

SEE ALSO

ex(1), edit(1)
Vi Quick Reference card, *An Introduction to Display Editing with Vi.*

AUTHOR

William Joy
Mark Horton added macros to *visual* mode.

BUGS

Software tabs using ^T work only immediately after the *autoindent.*

Left and right shifts on intelligent terminals don't make use of insert and delete character operations in the terminal.

The *wrapmargin* option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line won't be broken.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a : escape. To use these on a **:global** you must **Q** to *ex* command mode, execute them, and then reenter the screen editor with *vi* or *open*.

NAME

wait – await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with & have completed, and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the shell itself executes *wait*, without creating a new process.

EXAMPLE

```
wait
```

waits for all child processes to terminate.

SEE ALSO

sh(1).

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus can't be waited for.

NAME

wc — word count

SYNOPSIS

wc [**−lwc**] [names]

DESCRIPTION

Wc counts lines, words and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **−lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

EXAMPLE

```
wc filea fileb filec
```

reports the number of lines, words, and characters in each of the files.

NAME

what − identify SCCS files

SYNOPSIS

what files

DESCRIPTION

What searches the given files for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is **@(#)** at this printing) and prints out what follows until the first " , > , new-line, \ , or null character. For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

will print

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

What is intended to be used in conjunction with the SCCS command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

EXAMPLE

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
```

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

It's possible that an unintended occurrence of the pattern **@(#)** could be found just by chance, but this causes no harm in nearly all cases.

NAME

who − who is on the system

SYNOPSIS

who [−**uTlpdbrtas**] [file]

who am i

DESCRIPTION

Who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX System user. It examines the **/etc/utmp** file to obtain 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 the **am i** option identifies the invoking user.

Except for the default −**s** option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

−**u** This option lists information about those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in **/etc/inittab** (see *inittab*(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.

−**T** This option causes the *state* of the terminal line to be printed. The *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a − appears if it is not. **Root** can write to all lines having a + or a − in the *state* field. If a bad line is encountered, a ? is printed.

−**l** This option lists only those lines on which the system is waiting for someone to login. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field doesn't exist.

−**p** This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from **/etc/inittab** that spawned this process. See *inittab*(4).

-d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait*(2)), of the dead process. This can be useful in determining why a process terminated.

-b This option indicates the time and date of the last reboot.

-r This option indicates the current *run-level* of the *init* process. Following the run-level and date information are three fields which indicate the current state, the number of times that state was previously entered, and the previous state.

-t This option indicates the last change to the system clock (via the *date*(1) command) by **root**. See *su*(1).

-a This option processes **/etc/utmp** or the named *file* with all options turned on.

-s This option is the default and lists only the *name*, *line* and *time* fields.

EXAMPLE

```
who am i
```

reports the name under which you are currently logged in. This could be a name other than the original name under which you logged in, if the *su* command has been used.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), login(1), mesg(1), su(1), wait(2), inittab(4), utmp(4)
init(1M) in the *UniPlus*$^+$ *Administrator's Manual.*

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. At that point *write* writes **EOT** on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., **tty00**); otherwise, the first instance of the user found in **/etc/utmp** is assumed and the following message posted:

user is logged on more than one place.
You are connected to "*terminal*".
Other locations are:
terminal

Permission to write may be denied or granted by use of the *mesg*(1) command. Writing to others is normally allowed by default. Certain commands, in particular *nroff*(1) and *pr*(1) disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write inhibited terminal.

If the character ! is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., **(o)** for "over") so that the other person knows when to reply. The signal **(oo)** (for "over and out") is suggested when conversation is to be terminated.

EXAMPLE

write unisoft tty7

writes unisoft on terminal 7, unless messages have been refused with *mesg*(1).

FILES

/etc/utmp to find user
/bin/sh to execute !

SEE ALSO

mail(1), mesg(1), nroff(1), pr(1), sh(1), who(1).

DIAGNOSTICS

user not logged in if the person you are trying to *write* to is not logged in.

NAME

xargs − construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

Xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

Command, which may be a shell file, is searched for, using one's **$PATH**. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted: Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **−i** flag). Flags **−i**, **−l**, and **−n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **−l** vs. **−n**), the last flag has precedence. *Flag* values are:

−l*number* *Command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **−x** is forced.

−i*replstr* Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **−x** is also forced. {} is assumed for *replstr* if not specified.

−n*number* Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **−x** is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

−t Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

−p Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**−t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.

−x Causes *xargs* to terminate if any argument list would be greater than *size* characters; **−x** is forced by the options **−i** and **−l**. When neither of the options **−i**, **−l**, or **−n** are coded, the total length of all arguments must be within the *size* limit.

−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. **−e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). *Xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

Xargs will terminate if either it receives a return code of **−1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh*(1)) with an appropriate value to avoid accidentally returning with **−1**.

EXAMPLE

```
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 −1 ar r arch
ls | xargs −p −1 | 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.

DIAGNOSTICS

Self explanatory.

NAME

yacc − yet another compiler-compiler

SYNOPSIS

yacc [**−vdlt**] 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** option is used, this debugging code will be compiled by default. Independent of whether the **−t** option was used, the runtime debugging code is under the control of **YYDEBUG**, a pre-processor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

EXAMPLE

```
yacc file1.y
```

invokes *yacc* to process file "file1.y" in *yacc*-format.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.debug, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(1)
YACC − Yet Another Compiler Compiler.

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.