TEK SOFTWARE MANUAL

Part No. 070-5316-01
Product Group 65

UTek

COMMAND REFERENCE

VOLUME 1

First Printing NOV 1984
Revised SEP 1985

Tektronix®
COMMITTED TO EXCELLENCE

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California. We acknowledge the following individuals and institutions for their role in its development:

W. N. Joy
O. Babaouglu
R. S. Fabry
K. Sklower
Eric P. Allman
M. K. McKusick
E.Cooper
David Musher
S. J. Leffler

University of California at Berkeley
Department of Electrical Engineering and Computer Science

The MH Mail System is based on software developed by the Rand Corporation.

Portions of this document are based on the RCS Revision Control System, © 1982 Walter F. Tichy.

This documentation is for the use of our customers, and not for general sale.

Specifications subject to change.

Revision
INFORMATION

PRODUCT: 6000 Family Intelligent Graphics Workstation
This manual supports the following versions of this product: V2.2.

REV DATE	DESCRIPTION
NOV 1984	Original Issue
MAR 1985	Revised to support Version **2.1.**
SEP 1985	Revised to support Version **2.2** of Core, UTek/A, Pascal, and FORTRAN; and Version **2.3** of C and UTek/PS. Part number rolled to 070-5316-01.

Contents

NAME

@ — shell variable assignment (**csh** built-in)

SYNOPSIS

@ [*variable_operation*]

DESCRIPTION

With no arguments, **@** prints the names and values of all shell variables. With arguments, the assignment is done.

The normal form for *variable_operation* is *variable = expr*. The *variable* argument may either be a variable name or a reference to a member of a vector variable (the variable and element referenced must already exist). The *expr* argument is a numerical expression involving integers. See *csh(1csh)* for a complete description of expressions. In addition, operators such as *=, + =, and so forth, are available as in C. The spaces around the '=' are optional.

An alternate form of *variable_operation* involves the + + (increment) and — (decrement) operators. For example, the command "*@ i++*" increments the value of the variable *i*.

EXAMPLES

The following shell script prints the integers from 1 to 100.

```
#!/bin/csh -f
set a=1
while ($a <= 100)
        echo $a
        @ a++
end
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the message occurred.

CAVEATS

Spaces are required in separating elements of the expressions.

SEE ALSO

alias(1csh), bc(1), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dc(1), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), expr(1), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

abs — absolute value

SYNOPSIS

abs [—c*n*][*vector* ...]

DESCRIPTION

Output is the absolute value for each element of the input *vector(s)*. If no *vector* is given, the standard input is assumed.

OPTIONS

—c*n*

n is the number of output elements per line.

EXAMPLES

Outputs the absolute value of each element of A, three per line.

```
abs –c3 A
```

SEE ALSO

af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

adb — debugger

SYNOPSIS

adb [**—p** *prompt*] [**—w**] [**-l***dir*] [*objfil* [*corfil*]] [**-k**]

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 UTek 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.

Adb ignores QUIT; INTERRUPT causes return to the next **adb** command.

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.

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 prefixes **0o** and **0O** ("zero oh") force interpretation in octal radix; the prefixes **0t** and **0T** force interpretation in decimal radix; the prefixes **0x** and **0X** force interpretation in hexadecimal radix. Thus, **0o20** = **0t16** = **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. The radix must be set to decimal.

'cccc' The ASCII value of up to four characters. The backslash character (\) may be used to escape an apostrophe character (').

< 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. Some special provisions have been made for the names of the floating point registers on the National 32000 fpu. If the register name is specified starting with **F** (for example **F0**) it will be treated as a double floating register pair (for reading and writing). Entering the register name with an **f** (for example **f1**) will cause it to be treated as a single floating register.

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

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

routine.name
The address of the variable *name* in the specified C routine. Both *routine* and *name* are *symbols*. If *name* is omitted the value is the address of the most recently activated C stack frame corresponding to *routine*.

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

Monadic operators :

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

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

—exp
Integer negation.

~exp
Bitwise complement.

#exp
Logical negation.

Dyadic operators:

These 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 **?** (question mark).

=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:

o	2	Print two bytes in octal. All octal numbers output by **adb** are preceded by 0.
O	4	Print four 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 two bytes in hexadecimal.
X	4	Print four 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.
g 4 Print the floating constant as a 32 bit hex floating value.
G 8 Print the floating constant as a 64 bit hex double floating value.
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 ⟨CTRL—X⟩ and the delete character is printed as ⟨CTRL—?⟩
s *n* Print the addressed characters until a zero character is reached.
S *n* Print a string using the ⟨CTRL—X⟩ escape convention (see **C** above). The *n* is the length of the string including its zero terminator.
Y 4 Print four bytes in date format (see *ctime(3c)*).
i n Print as machine instructions. The *n* is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
a 0 Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below:

/ Local or global data symbol.
? Local or global text symbol.
= Local or 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, it tabs to the next appropriate tab stop. For example, **8t** moves to the next eight–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.

[ ] Right and left square brackets. Set memory breakpoint at address and break if the memory **read/write** has occured. If **r** appears inside the brackets, as in **[r]**, then the memory breakpoint is set at the address specified and the break occurs on a memory **read**. If **w** appears inside the brackets, then the

memory breakpoint is set at the address and the break occurs on a memory **write**. If **d** appears inside the brackets, then the breakpoint (specified by address) is **deleted**.

[**?**/]**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 four bytes at a time instead of two. 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 two–byte *value* into the addressed location. If the command is **W**, write four 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. For example, **/m?** will cause / to refer to *objfil*.

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

! A shell (*/bin/sh*) 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 causes 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 the contents of the frame—pointer register. 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.

q Exit from **adb**.

v Print all nonzero variables in octal.

m Print the address map.

t Toggle the virtual to physical trace output. Only useful if **–k** command line option is also specified.

n Toggle the National–32000/Compiler disassembler display format (default is National–32000).

:*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 *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.

c*s* The subprocess is continued with signal *s* (see *sigvec(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* Same 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.

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 *filename address* corresponding to a written *address* is calculated as follows:

```
b1 ≤ address 〈 e1 => filename address = address + f1-b1,
```

or

```
b2 ≤ address 〈 e2 => filename address = address + f2-b2
```

If not in this form, the requested *address* is not legal. In some cases, such as 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.

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 or 0413).
s The stack segment size.
t The text segment size.

OPTIONS

—p Makes **adb** interpret the following string as the prompt to use when it is ready to accept commands from your terminal.

—w
Both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using **adb**.

—l Specifies a directory where files to be read with $< or $<< (see below) will be sought; the default is */usr/lib/adb*.

—k Specifies that *corfil* represents physical memory of a UTek kernel (either */dev/mem* or the *vmcore.?* file generated by *savecore(8)*). Enables virtual to physical address translation when looking in *corfil* .

FILES

a.out	Default binary filename.
core	Default core image filename.

DIAGNOSTICS

adb
This is given when there is no current command or format.

This command comments about inaccessible files, syntax errors, abnormal termination of commands, and so forth.

Exit status is 0, unless the last command failed or returned nonzero status.

CAVEATS

Since no shell is invoked to interpret the arguments of the **:r** command, the customary wild-card and variable expansions cannot occur.

SEE ALSO

cc(1), sdb(1), ptrace(2), a.out(5), core(5), savecore(8).

NAME

af — arithmetic function

SYNOPSIS

af [**—c***n*] [**—t**] [**—v**] *expression ...*

DESCRIPTION

Output is the result of executing the *expression*(s) once per complete set of input values. Input comes from vectors specified in *expression*.

Expression operands are:

Vectors:
filenames with the restriction that they must begin with a letter and be composed only from letters, digits, '_', and '.'. The first unknown filename (one not in the current directory) references the standard input.

Functions:
the name of a command followed by the command arguments in parentheses. Arguments are written as on the command line, separated by spaces.

Constants:
floating point and integer (but not E notation).

Expression operators are, in order of decreasing precedence:

'v	next value from vector *v*.
x^y, *–x*	*x* raised to the *y* power, negation of *x*; both associate right to left.
*x*y*, *x/y*, *x%y*	*x* multiplied by, divided by, modulo *y*, respectively; all associate left to right.
x+y, *x–y*	*x* plus, minus *y*, respectively; both associate left to right.
x,y	output the value of *x* followed by the value of *y*; associates from left to right

Parentheses may be used to alter precedence. Because many of the operator characters are special to the shell it is good practice to surround *expressions* in quotes.

OPTIONS

—c*n*
n elements per line in the output.

—t output is titled from the vector on the standard input.

—v verbose mode, function expansions are echoed.

EXAMPLES

```
af "3+4*5"
```

yields 23.

```
af "A,'A,A+'A,B"
```

yields a four column matrix with columns of
1) odd elements from A,
2) even elements from A,
3) sum of adjacent odd and even elements, and
4) elements from B.

```
af "sin(A)^2"
```

yields the square of the sine of the elements of A.

SEE ALSO

abs(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

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**] *filenames...*

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 — (a dash)), and named files (note that SCCS file names must begin with the **s.** character). 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. If a name of — (a dash) 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.

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.

The last component of all SCCS filenames must be of the form **s.***filename*. 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.***filename*, (see *get(1sccs)*), 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.***filename*), which is used to prevent simultaneous updates to the SCCS file by different users. See *get(1sccs)* for further information.

OPTIONS

—**a***login*
A *login* name, or numerical UTek 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.

—**f***flag[flag—val]*
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:

FLAGS

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

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

—**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.

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

—**f***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.

—i Causes the "No id keywords (ge6)" message issued by *get(1sccs)* or *delta(1sccs)* 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(1sccs)*) are found in the text retrieved or stored in the SCCS file.

—j Allows concurrent *get(1sccs)* 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 be "unlocked". See the **—f** keyletter for a description of the **—l** flag and the syntax of a *list*.

—m*mod*
Module name of the SCCS file substituted for all occurrences of the **%M%** keyword in SCCS file text retrieved by *get(1sccs)*. If the **—m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s.** removed.

—n Causes *delta(1sccs)* to create a null delta in each of those releases (if any) being skipped when a delta is made in a *new* release (for example, 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(1sccs)*.

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

—v*[pgm]*
Causes *delta(1sccs)* 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(1sccs)*). (If this flag is set when creating an SCCS file, the **—m** keyletter must also be used even if its value is null).

MORE OPTIONS

—h Causes **admin** to check the structure of the SCCS file (see *sccsfile(5sccs)*), 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.

i*[name]*

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

—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(1sccs)*. 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.

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

—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 filename must also be supplied. In the case of existing SCCS files: (1) a **—t** keyletter without a filename causes removal of descriptive text (if any) currently in the SCCS file, and (2) a **—t** keyletter with a filename causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

—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(1sccs)*. 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 (for example, a new SCCS file is being created).

—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.

DIAGNOSTICS

Use *sccshelp(1sccs)* for explanations.

CAVEATS

Non–SCCS files and unreadable files are silently ignored.

SEE ALSO

delta(1sccs), ed(1), get(1sccs), sccshelp(1sccs), prs(1sccs), what(1sccs), sccsfile(5sccs).

NAME

alias, unalias — alias substitutions (**csh** built–in)

SYNOPSIS

alias [*name* [*wordlist*]]
unalias *pattern*

DESCRIPTION

The shell maintains a list of aliases which can be established, displayed, and modified by the **alias** and **unalias** commands. With no arguments, **alias** prints the names and wordlists for all aliases set. With a *name,* the wordlist for that name is printed. With a *name* and a *wordlist,* the wordlist is subjected to command and filename substitution and assigned to the name as an alias. The **unalias** command deletes any aliases that match the given *pattern.* The command **unalias** * deletes all aliases.

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 you can **alias print 'pr \!* ¦ lpr'** to make a command which **pr**'s its arguments to the line printer.

EXAMPLES

The following command causes the command "ls –lR" to be substituted for occurrences of the word "list" when it occurs in a command position.

```
alias list ls -lR
```

This command will delete any two–character aliases that end with the letter 'e'.

```
unalias ?e
```

CAVEATS

Aliases can not contain the names "alias" or "unalias".

If an alias contains it's own name in backquotes, an infinite loop may occur, which will crash the shell. An example of one of these is

```
alias ls 'echo `ls`'
```

Aliases can not always be used in place of simple commands, such as in simple **if** statements.

When one command of a multi–command alias is suspended, the other commands are forgotten by the shell. If all commands of a multi–command alias need to be executed without being affected by suspension, the alias should be surrounded by parentheses, as in the following.

```
alias change '(co -l \!* ; vi \!* ; ci -u \!* )'
```

SEE ALSO

@(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

apropos — whatis database keyword search

SYNOPSIS

apropos [**—a**] [**—d**] [**—f** *file*] [**—i** *section*] [**—p**] [**—s** *section*] [**—v**] [*keyword ...*]

DESCRIPTION

Apropos searches whatis database files, as described in *whatis(5man),* for given keywords and formats the matching database entries.

Either keywords or a section must be specified. If a section is specified with the **—s** option, only entries which match the section will be considered. If a section is specified with the **—i** option, only entries which do not match the section will be considered. The *section* argument consists of a section number and optional subsection text, such as '1' or '1mh'. The section number can be a '+', which means that any section number is allowed, such as '+' (meaning 1–8) or '+mh' (meaning any section number from 1 to 8 followed by the subsection text 'mh'). Also, the subsection text can be replaced by a '+', as in '1+', which matches any section that begins with a 1 (including '1' alone). The argument '++' is not valid.

Keywords and sections are matched without respect to case. For example, the keyword 'file' will match 'file', or 'File', or any other combination of upper and lower case characters.

If the **—f** option is not given, the file /usr/lib/man/directories is read to get the locations of the database files that correspond to directories listed in the PATH environment variable, and the file $HOME/.manrc is read to get the locations of personal manual page directories.

The normal format for a line is

page (*section*⟨tabs⟩ – *description*

where *page* is the manual page name, *section* is the manual page section, and *description* is the text from the NAME section of the manual page found after the '–'. The ⟨tabs⟩ field is enough tabs to fill out the first 24 positions on the line.

OPTIONS

—a Print only entries that contain *all* keyword entries. Normally, only one keyword has to be found for the entry to match.

—d Search only the description field. Normally, the page and description fields of the entry are searched.

—f *file*
Search the named file as a whatis database instead of using the defaults.

—i *section*
Print only entries whose section field does not match the given section specifier.

—p Print only the manual page name and the section for matching entries.

—s *section*
Print only entries whose section field matches the given section specifier.

—v Verbose. Before the matching entries for a given whatis database are printed, print the name of the database with a short description. In the case of databases that correspond to elements of the PATH variable, the path element is noted instead.

EXAMPLES

The following invocation prints the descriptions for all manual pages in section 3 and all of its subsections.

```
apropos -s 3+
```

This invocation prints the names and sections of all manual pages whose name or description contains the words 'read' or 'write', preceding the entries from each database with a description of the database.

```
apropos -pv read write
```

This invocation prints the names and sections of all manual pages whose description contains all of the words 'execute', 'command', and 'time' in the description field.

```
apropos -a -d execute command time
```

FILES

/usr/lib/man/directories	Manual page search directory information.
$HOME/.manrc	Searched for "personal" manual page directory names.
whatis	The name of the whatis database file.

VARIABLES

PATH	The user's execution path.
HOME	The user's home directory.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

buildif(1man), help(1man), makewhatis(1man), man(1man), more(1), section(1man), whatis(1man), man(5man), manindex(5man), whatis(5man), catman(8man).

NAME

ar — archive and library maintainer

SYNOPSIS

ar (dmpqrtx)[abcfilnosuv [*member*]] *afile* [*name*...]

DESCRIPTION

Ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the loader. It can be used, though, for any similar purpose.

Key is one character from the set **dmpqrtx**, optionally concatenated with one or more of **abcfilnosuv** (flag). In cases where the *flag* is a positioning character, such as **a** or **b**, the *member* argument is the name of the archive member which is used as a relative position. The *afile* argument is the archive file. The *name* arguments are constituent files in the archive file.

If the key given is **d, m, q,** or **r,** or if the **s** flag is given, the symbol definition table is added or modified by executing *ranlib(1)*. This can be turned off with the **f** flag.

It is very important to note that the archive format used on this system is not the same as on other systems. See the manual page for *ar(5)* for information on the format, and see the second example below for the method of converting archives to a portable format.

OPTIONS

Keys:

d Delete the named files from the archive file.

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

p Print the named files in the archive.

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. The **q** key is useful only to avoid quadratic behavior when creating a large archive piece–by–piece.

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

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.

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. Normally the 'last-modified' date of each extracted file is the date when it is extracted. However, if **o** is used, the 'last-modified' date is reset to the date recorded in the archive.

Position Names and Flags:

a Position after. The name following the options list is the name of the archive member after which the given file is to be positioned. See the descriptions for the keys **m** and **r,** which are the only options for which the **a** position name is valid.

b Position before. The name following the options list is the name of the archive member before which the given file is to be positioned. See the descriptions for the keys **m** and **r,** which are the only options for which the **b** position name is valid.

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

f Prevent symbol definition table creation. This is the opposite of the **s** flag. If both **f** and **s** are given, the **s** is ignored.

i Insert before. This is a synonym for the **b** position name.

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

n Normal format. This version of **ar** does not truncate names longer that 15 characters for insertion into the archive. In order to use an archive with software which does not support this archive format, the **n** flag must be used to force truncation of names.

o Preserve original date. This flag is used with the key **x** and causes extracted files to have the last-modified date reset to the date recorded in the archive file.

s Create or update symbol definition table. This flag causes the command **ranlib** to be executed with the archive name as the argument.

u Replace updated files. This flag is used with the **r** key, and causes only those files with modification dates newer than the corresponding member in the archive.

v Verbose. With the verbose flag, **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. Otherwise, each file is preceded by the key letter of the operation done. For example, **r** — *filename* (for replace) and **x** — *filename* (for extract).

EXAMPLES

The following invocation of this command,

```
ar x utils move copy remove
```

extracts the files *move, copy,* and *remove* from the archive file *utils,* and places them in the current working directory.

The following shell script converts a long format archive file to a normal format archive:

```
#!/bin/sh -x
#
# Convert long format archive to normal format.
#
if test $# -ne 1
then
        echo "$0 : usage : $0 file"
        exit 1
fi
Magic=`head -1 $1`
if test "!<arch>" = "$Magic"
then
        echo "$0 : The file $1 is already in normal format."
        exit 1
fi

if test "!<ARCH>" != "$Magic"
then
        echo "$0 : The file $1 is not an archive file."
        exit 1
fi

#
# Build a temporary directory to hold the members.
#
err=1
trap "rm -rf tmp.$$;" exit $err 0 1 2 3 15
mkdir tmp.$$
if test $? -ne 0
then
        exit 1
fi

cd tmp.$$
ar x ../"$1"
```

```
ar crs tmp.a *
mv tmp.a ../"$1"
cd ..
err=0
```

FILES

*/tmp/v**	Temporaries

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

The *last-modified* date of a file will not be altered by the **o** option if the user is not the owner of the extracted file or is not the superuser.

The **s** flag causes **ranlib** to be executed using the user's execution path. In addition, **ranlib** executes **ar** using the user's execution path.

SEE ALSO

cpio(1), file(1), ld(1), make(1), nm(1), ranlib(1), fgetarhdr(3c), getarhdr(3c), ar(5).

NAME

arcv — convert archives to new format

SYNOPSIS

arcv [**—t**] *filename ...*

DESCRIPTION

Arcv converts archive files (see *ar (1) , ar (5))* from 32v and Third Berkeley editions to a new portable format. The conversion is done in place, and the command refuses to alter a file not in old archive format.

Old archives are marked with a magic number of 0177545 at the start; new archives have a first line "**!<arch>**".

OPTIONS

—t Put temporary file in */usr/tmp* instead of */tmp.*

EXAMPLES

The following example converts the file *old.a* to the new archive format.

```
arcv old.a
```

FILES

*/tmp/arc**	Temporary copy of the archive
*/usr/tmp/arc**	Temporary copy of the archive (with **—t** option)

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

ar(1), ar(5).

NAME

assign, deassign — assign or deassign devices in a class

SYNOPSIS

assign [**—f**] [**—r**] class ...
deassign class ...

DESCRIPTION

The **assign** and **deassign** commands use the file */etc/assign.classes* to determine what devices are assignable and what class(es) they belong to. Assignable devices are divided into classes (e.g. mag tape drive 0) so that all versions of a device can be assigned together (raw and cooked versions for example). These classes, in turn, may be grouped into generic classes (e.g. any mag tape drive). These groupings are set up by the system administrator, and thus are installation dependant.

Assign changes the owner of the device(s) in the requested *class* to the current UID. If the requested class is a generic one, the first specific class found that is available is used. The device(s) assume protection mode of read and write by owner only. If more than one class is listed, the first one that contains an assignable device is used and the remainder are ignored. This allows the user to specify the order that classes will be checked.

The name of the specific class that was assigned is typed on standard output. If no devices are available in the class(es) desired, an error message is typed on standard error, and the program exits with return code 1, unless the **—r** option is used.

Deassign undoes the work of **assign** by releasing the device to the assignable pool. The special form **deassign all** will deassign all devices you have.

All devices are considered available to the superuser. If the log file does not exist, logging is ignored. The owner of devices that are free for assignment is the owner of the file */etc/assign.classes*.

OPTIONS

—f Does not prompt whether to deassign when used in conjunction with the **—r** flag. This allows *assign* **—r** to be used in a shell script.

—r Reassigns the devices of a requested class after determining that no requested class is available without overriding an assignment. Mail is sent to the user with the previous assignment. Unless used with the **—f** flag, a prompt is issued to determine whether to override the previous assignment of the device(s).

EXAMPLES

assign mt1 mt0

would try to assign class *mt1*, and would try for *mt0* only if *mt1* is unavailable.

Typical usage might be (using the Bourne shell):

mt = 'assign mt'
while test ! "${mt}"
do sleep 120; mt = 'assign mt'; done

do whatever needs doing on the mag tape using device names:

/dev/${mt}0 for 800 BPI cooked
/dev/${mt}1 for 1600 BPI cooked
/dev/r${mt}0 for 800 BPI raw
/dev/r${mt}1 for 1600 BPI raw
/dev/n${mt}0 for 800 BPI cooked, no rewind on close
/dev/n${mt}1 for 1600 BPI cooked, no rewind on close
/dev/nr${mt}0 for 800 BPI raw, no rewind on close
/dev/nr${mt}1 for 1600 BPI raw, no rewind on close

This assumes that the system administrator has set up a generic class name in */etc/assign.classes* of *mt* and a specific class names of form *mt0*, *mt1*, *mt2* ... and that the tape versions of the tape drives are named /dev/mt00, /dev/mt01, /dev/rmt00 ... The shell variable *mt* will have the name of the assigned tape drive (e.g. *mt0*, *mt1*, *mt2* ...)

FILES

/etc/assign.classes	System file containing class specifications
/usr/adm/devicelog	Log file for device usage

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	No available class was found.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

[P_ERR] A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

assign.classes(5).

NAME

at — execute commands at a later time

SYNOPSIS

at [**-m**] [**-v**] *time* [*day*] [*filename*]

DESCRIPTION

At squirrels away a copy of the named file (standard input default) to be used as input to *sh (1sh)* (or *csh (1csh)* if you normally use it) at a specified later time. A **cd** command to the current directory is inserted at the beginning, followed by assignments to all environment variables (excepting the variable TERMCAP, which is useless in this context.) When the script is run, it uses the user and group ID of the creator of the copy file.

The *time* is one to four digits, with the optional suffixes **A**, **P**, **N**, or **M** for AM, PM, noon or midnight, respectively. Semicolons separating hours and minutes are allowed. 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 a month name followed by a day number, or 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.

At programs are executed by periodic execution of the command **/usr/lib/atrun** from *cron (8)*. The granularity of **at** depends upon how often **atrun** is executed.

Standard output or error output is lost unless redirected.

OPTIONS

-m Send mail notification when *at* job is run.

-v Verbose. Prints the name of the file containing your *at* job. The filename contains the date at which the job is scheduled to run.

EXAMPLES

Examples of legitimate commands are

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

where *cmdfile* contains *sh (csh)* commands.

FILES

/usr/lib/atrun	Executor (run by *cron(8)*)
*/usr/spool/at/yy.ddd.hhhh.**	Activity for year *yy*, day *dd*, hour *hhhh*
lasttimedone	Last *hhhh*
past	Activities in progress

VARIABLES

SHELL	The user's login shell. If it is */bin/csh*, that shell is used to execute commands.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

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.

Only twenty jobs may be scheduled to run in a single minute. All subsequent jobs will be rescheduled for the next minute that has open slots.

SEE ALSO

calendar(1), csh(1csh), pwd(1), sh(1sh), sleep(1), cron(8).

NAME

awk — pattern scanning and processing language

SYNOPSIS

awk [**—Fc**] [*prog*] [*filename* . . .]

DESCRIPTION

Awk scans each input *filename* 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 *filename* matches the pattern. The set of patterns may appear literally as *prog,* or in a file specified as **—f** *filename.*

Files are read in order; if there are no files, the standard input is read. The filename — (a dash) 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**, *vide infra.*) 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, newlines, 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 *>filename* 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 (1mh)* 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(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 a tilde (~) (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 newline); and *OFMT*, the output format for numbers (default "%.6g").

OPTIONS

—F*c*

Designates the single character *c* as the field separator. The character may be a separate argument, as in **—F :** (where the field separator character is a colon). The character *t* specifies that the field separator is a tab.

—f*filename*

The set of patterns are contained in *filename*. The *filename* may be a separate argument.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

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

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

CAVEATS

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 quotation marks (“”) to it.

When the functions *log(), exp(),* and *sqrt()* are executed with values that are not in the domain of these functions, or would return values that are out of the machine's range, error messages are printed and the program terminates.

SEE ALSO

comm(1), cut(1), egrep(1), fgrep(1), grep(1), join(1), lex(1), look(1), paste(1), regcmp(1), sed(1), sort(1), uniq(1), exp(3m), printf (3s).

NAME

bar — build a bar chart

SYNOPSIS

bar [**—a**][**—b**][**—f**][**—g**][**—r***i*][**—w***i*][**—x***f*][**—xa**][**—y***f*][**—ya**][**—yl***f*][**—yh***f*][*vector ...*]

DESCRIPTION

Output is a GPS that describes a bar chart display. Input is a *vector* of counts that describes the y–axis. By default, x–axis will be labelled with integers beginning at 1; for other labels, see *label.* If no *vector* is given, the standard input is assumed.

OPTIONS

—a Suppress axes.

—b Plot bar chart with bold weight lines, otherwise use medium.

—f Do not build a frame around plot area.

—g Suppress background grid.

—r*n*
Put the bar chart in GPS region *n*, where *n* is between 1 and 25 inclusive.

—w*n*
n is the ratio of the bar width to center–to–center spacing expressed as a percentage. Default is 50, giving equal bar width and bar space.

—x*n* (**y***n*)
Position the bar chart in the GPS universe with x–origin (y–origin) at *n*.

—xa (ya)
Do not label x–axis (y–axis).

—yl*n*
n is the y–axis low tick value.

—yh*n*
n is the y–axis high tick value.

EXAMPLES

Outputs the bar chart described by vector A, located in region 10 of the GPS universe, with no x–axis labels. Bar width is 75% of the center–to–center spacing.

```
bar –r10,xa,w75 A
```

SEE ALSO

abs(1g), af(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

basename — strip filename affixes

SYNOPSIS

basename *string* [*suffix*]

DESCRIPTION

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

EXAMPLES

This example compiles the named file and moves the output to *cat* in the current directory:

```
cc /usr/src/cmd/cat.c
mv a.out `basename /usr/src/cmd/cat.c .c`
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.

SEE ALSO

sh(1sh).

NAME

basic — interactive Basic language

SYNOPSIS

basic [*filename.d filename.a*] ...

basic.o [*/usr/lib/basic/gpib.d /usr/lib/basic/gpib.a*] [*/usr/lib/basic/xio.d /usr/lib/basic/xio.a*] [*filename.d filename.a*] ...

DESCRIPTION

Basic is the interactive Basic language programming environment. It resembles conventional Basic interpreters, but executes your program by compiling it to machine code rather than by interpreting it.

The language is a superset of the proposed ANSI standard, and is described in detail in the references listed below.

The **Basic** command is a shell script, while **basic.o** is the executable program. The shell script performs a **stty(1)** command to disable the "dsusp" and "flush" control characters and to set the "lnext" character to control–underscore, which lets you use the control–Y, control–O, and control–V keys for editing. It then executes **basic.o** with parameters */usr/lib/basic/gpib.d /usr/lib/basic/gpib.a /usr/lib/basic/xio.d /usr/lib/basic/xio.a,* which causes the GPIB and XIO extensions to be installed and made available. Any parameters to the **basic** command are expected to be pairs of descriptor and archive files (extensions ".d" and ".a") and are passed along to **basic.o.**

If you do not need the GPIB or XIO extensions, you can make Basic start up faster by changing the control characters yourself and invoking **basic.o.**

After Basic has installed any extensions, and before it prompts for the first command from the keyboard, it looks for a file in your current directory (*not* your home directory) named *.basic.profile.* If it exists, Basic executes the command **SCRIPT** *.basic.profile.*

If nobody at your site uses one or both extensions, your system administrator can make Basic start up more quickly by editing the file */bin/basic* and removing some of the arguments to the command **/bin/basic.o.**

It is possible to change the binding of the editor command keys. This is done by setting the KEYS environment variable to a string which is exactly 32 characters long. The remapping works as follows: for each "i" from 0 to 31 (decimal), the key with ASCII code "i" is mapped to the editor function normally invoked by the control character control–X, where X is the character at position "i" in the KEYS variable. For example, to make control–A mean "move left one character" (which is normally invoked by control–V), the character in position 1 (the ASCII code for control–A) of the string is set to the letter V, with the **sh(1)** command

```
KEYS="@VBCDEFGHIJKLMNOPQRSTUVWXYZ[]^_"; export KEYS.
```

Remember that the first position in the KEYS string is number 0, which corresponds to the character control–@ (the NULL character).

There are two exceptions to this remapping. Since Unix translates the key control–M (carriage return) into control–J (newline), control–M is always bound to the function to which control–J is bound, regardless of the character in position 13 of the string. And there is no way to remap the RUBOUT (DELETE) character.

EXAMPLES

Normally, all you need to do to start Basic is to enter

```
basic
```

If you want Basic to start up quickly, and you don't use the GPIB extension (built–in facilities whose names begin with "G_"), and you don't use the XIO extension (built–in facilities whose names begin with "X_"), you can use

```
/bin/stty dsusp u flush u lnext "^_"
basic.o
```

FILES

.basic.profile	SCRIPTed when Basic starts up.
BASICnnnnnn	(nnnnnn is the number of the Basic process) Your program is saved in this file when executing a VI command, and also when Basic terminates due to an internal system error.
/bin/basic	The shell script which invokes /bin/basic.o.
/bin/basic.o	The executable Basic programming environment.
/etc/termcap	Examined for a terminal capability description if the TERMCAP variable is not set.
/usr/gks/filter/tekterm	Driver for 4105, 4106, 4107, and 4109 terminals
/usr/gks/fontlist	List of available character fonts
/usr/gks/wdf/4105	Workstation Descriptor File for 4105 terminals running Basic
/usr/gks/wdf/4106	Workstation Descriptor File for 4106 terminals running Basic
/usr/gks/wdf/4107	Workstation Descriptor File for 4107 terminals running Basic
/usr/gks/wdf/4109	Workstation Descriptor File for 4109 terminals running Basic

VARIABLES

BASIC_DUMP	If defined, Basic stops and dumps core when an internal system error occurs. Normally Basic writes your program into BASICnnnnn and exits.
EDITOR	The editor to use for a VI command (default vi).
KEYS	Editor key binding map.
PROMPT	Command prompt, normally '› '. Can include newlines.
STROKESIZE	Maximum number of points that can be read by a MAT LOCATE or MAT GET statement (default 100).
TERM	The type of terminal being used.
TERMCAP	The file containing the terminal capability entry, or the entry itself (default /etc/termcap).

REFERENCES

ANSI BASIC User's Guide
ANSI BASIC Keyword Dictionary
ANSI BASIC Quick Reference

NAME

batch — MDQS batch command spooler

SYNOPSIS

batch [**—a** *atime*] [**—m**] [**—o** *outfile*] [**—p** *priority*] [**—q** *queue*] [**—s** *shell*] [**—u** *useraddr*] [**—v**] [**—R** *ruser*] [*commandfile*]

DESCRIPTION

Batch is used to queue command files to be run by the Multi-Device Queueing System (MDQS). The command files are basically shellfiles. They can contain a single command or a series of commands. If the *commandfile* argument is omitted, the commands are read from the standard input. The batch command tries to record the environment that the user was in when the batch command was invoked, including exported shell variables and the current working directory.

Batch is part of the MDQS. Once the request has been queued, it can be modified or deleted using the **qmod** program, and the current status determined using the **qstat** program.

OPTIONS

—a *atime*
: Do not start this request until after *atime*. For the format of time specifications, see *getdate(5mdqs)*.

—m
: Cause a message to be generated upon the completion of the batch request and sent via *mail(1mh)* to the requesting user.

—o *outfile*
: Causes the standard output and error output (filedescriptors 1 and 2) of the batch job to be redirected to the named file.

—p *priority*
: Allows one to give a request a priority other than the default. The default priority is in the middle of the range 0–10. Only the superuser, the mdqs user or a member of the systems groups may request a priority of 0.

—q *queue*
: Allows one to queue a request to a queue other than the default. Queueing a request to other than a batch queue will give unpredictable results.

—R *ruser*
: When a batch request is to execute on a remote host, MDQS normally requires the requesting local user name to exist on the remote host with *rsh(1n)* privileges. This option allows one to specify another user on the remote host for *rsh(1n)* privileges.

—s *shell*
Causes the program *shell* to be used as the shell to interpret the command file instead of the default. When the commands are read from standard input the default shell is the one specified by the environmental variable *SHELL,* if it is available. Otherwise the default shell is *(/bin/sh).*

—u *useraddr*
Overrides the default notification address with the address *useraddr.* This address should be a valid address to your mailsystem. The default notification address is the name of the invoker on the current host.

—v Normally the batch command does its work silently. The v (verbose) option makes batch acknowledge that the request has been queued and prints the request number.

FILES

/usr/lib/mdqs/queue2	Second stage queuer
/etc/mdqsd	MDQS daemon
/usr/spool/q	Top of spooling directory tree
/usr/lib/mdqs/shserver	Program called by the daemon to run shell
/etc/qconf	Configuration information for MDQS

VARIABLES

SHELL	The user's login shell.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

qstat(1mdqs), qmod(1mdqs), getdate(5mdqs), mdqsd(8mdqs), rsh(1n).

NAME

bc — arbitrary-precision arithmetic language

SYNOPSIS

bc [**—c**] [**—f**] [**—l**] [*filename ...*]

DESCRIPTION

Bc is an interactive processor for a language which resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input, unless the **—f** option is given.

The syntax for **bc** programs is as follows: **L** means letters 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 newlines 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, as a function, and as 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.

Error messages are always sent to the standard output to be printed by **dc.** If the **—c** option is given, error messages about running out of space are also sent to standard error.

OPTIONS

—c Compile only — does not invoke **dc**. **Dc** input sent to standard output.

—f Don't read from standard input.

—l Uses an arbitrary precision math library.

EXAMPLES

The following defines a function to compute an approximate value of the exponential function:

```
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
}
}
```

This next example prints approximate values of the exponential function of the first ten integers:

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

FILES

/usr/lib/lib.b	mathematical library

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

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

If the scale factor is greater than 63, the error in multiplication can be as big as 199.

SEE ALSO

dc(1).

NAME

bdiff — big diff

SYNOPSIS

bdiff *filename1 filename2* [*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 causing it to fail. If *filename1* (*filename2*) is a dash (—), 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.

OPTIONS

—s Specifies that no diagnostics are to be printed by **bdiff**.

FILES

/tmp/bd?????	Temporary file for performing the **diff** command.

DIAGNOSTICS

Use *sccshelp (1sccs)* for explanations.

SEE ALSO

diff(1).

NAME

bel — send bel character to terminal

SYNOPSIS

bel

DESCRIPTION

Bel causes most terminals to sound an audible tone, a useful nonvisual signal.

SEE ALSO

abs(1g), af(1g), bar(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

fg, bg — foreground and background jobs (**csh** *built-in)*

SYNOPSIS

fg [**%***job*...]
or
%*job*
bg [**%***job*...]
or
%*job***&**

DESCRIPTION

The command **fg** brings a stopped or backgrounded job into the foreground, meaning that the job becomes attached to the terminal. The command **bg** puts a stopped job in the background, meaning that the job becomes detached from the terminal. The syntax for *job* is described in the manual page *jobs(1csh).* If no job argument is given, the current job is used (this job is marked by a '+' in the listing given by **jobs .)**

The syntax **%***job* is an alternate for **fg,** and **%***job***&** is an alternate for **bg.**

EXAMPLES

A common way to follow a *make(1)* command's progress is to redirect the output to a file and use the command **tail –f***file* to view the output. Usually, the *tostop* mode of the terminal is used (see *stty(1)), so that the command stops* is ready. To do this, the command

```
tail -f make_output &
```

is executed. When it is ready to send output to the terminal, a message like:

```
[2] + Stopped (tty output) tail -f make_output
```

is printed. At this point, any of the following commands can be used to bring the command into the foreground for viewing of the output:

```
fg
fg %2
fg %tail
%2
%tail
```

When the output has stopped or slowed to the point that it would be best to wait for more output to be built up, the character ^Z (control–z) can be typed to stop the command. At this point, the command can be put back into the background by using one of the following commands:

```
bg
bg %2
bg %tail
%2 &
%tail &
```

DIAGNOSTICS

command: No such job.
The job named (either implicitly or explicitly) is not a job.

command: Ambiguous.
There is more than one job that matches the pattern given.

CAVEATS

When *tostop* mode is set and a lot of jobs are running, it can be very confusing, since the job that was last announced as being the current job may no longer be so.

Backgrounding an already backgrounded job will cause the job to be marked so that it does not become the current job again automatically.

SEE ALSO

@(1csh), alias(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), signal(3c).

NAME

break — exit from loop (**csh** built–in)

SYNOPSIS

break

DESCRIPTION

The **break** command causes execution to resume after the *end* statement of the enclosing **while** or **foreach** loop. The remaining commands on the current line are executed, thus, multi–level breaks may be implemented by putting multiple **break** commands on the same line.

EXAMPLES

The following example shows a *while* statement which is used to read data from the standard input and process the data using the (undefined) command 'process'. The **break** is used to exit the loop when the string "end" is read.

```
while (1)
        set x="$<"
        if ("$x" == "end") then
                break
        endif
        process "$x"
end
```

DIAGNOSTICS

break: Too many arguments.
The **break** command does not take arguments, but arguments were given.

break: Not in while/foreach.
A **break** command was found outside of a *while* or *foreach* statement.

RETURN VALUE

[NO_ERRS] Command completed without error.

[1] An error of the type described in the message occurred.

CAVEATS

Unlike the *sh(1sh)* built–in **break,** this command does not take an argument.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1sh), cd(1csh), chdir(1csh), continue(1csh), continue(1sh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

break — exit from a **for** or **while** loop (*sh* built-in)

SYNOPSIS

break [*n*]

DESCRIPTION

Break exits from the enclosing **for** or **while** loop. If *n* is specified, break *n* levels.

EXAMPLES

The following shell script reads lines from the standard input. If the first word is "end", the rest of the line is printed and the shell script terminates. The loop is left by using **break.**

```
#!/bin/sh
while read f g
do
        if test "$f" = "end"
        then
                break
        fi
done
echo $g
```

RETURN VALUE

[NO_ERRS] Command completed without error.

SEE ALSO

break(1csh), cd(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), execve(2).

NAME

bucket — generate buckets and counts

SYNOPSIS

bucket [—**a***n*] [—**c***n*] [—**F***vector*] [—**h***n*] [—**i***n*] [—**l***n*] [—**n***n*] [*vector ...*]

DESCRIPTION

Output is a vector with odd values being bucket limits (in parentheses) and even values being the number of elements from the input within the limits. Input vectors are assumed to be sorted. If no input *vector(s)* are given, the standard input is assumed. Unless otherwise specified, bucket limits are generated based on the input data and the rule:

num buckets = 1 + log2(num elements).

OPTIONS

—**a***n*
: choose limits such that *n* is the average count per bucket.

—**c***n*
: *n* elements per line in the output.

—**F***vector*
: take limit values from *vector*.

—**h***n*
: *n* is the highest limit.

—**i***n*
: *n* is the interval between limits.

—**l***n*
: *n* is the lowest limit.

—**n***n*
: *n* is the number of buckets.

EXAMPLES

The following command outputs limits and counts for the elements of A, where the lowest limit is –5 and the average bucket count is 12.

```
bucket -a12,l-5 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

buildif — build index format data

SYNOPSIS

/usr/lib/buildif [**—f**] *filename ...*

DESCRIPTION

Buildif reads each of the named files, which are expected to be formatted manual pages, locates the sections NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXAMPLES, FILES, DIAGNOSTICS, VARIABLES, RETURN VALUE, CAVEATS, SEE ALSO, and REFERENCES, and places a table as described in *manindex(5man)* at the end of the file. The NAME section is required, and all others are optional. This table is used by the commands *help(1man)* and *section(1man)* to display sections of the manual entry.

This command may be executed automatically by *catman(8man)*. See that manual entry to find out how this is done.

Unless the **—f** option is given, a manual entry that already has an index format table will not be processed.

OPTIONS

—f Force. If the file already has an index format table, remove the table and build a new one.

EXAMPLES

The following example will add a new index format table to the file */usr/man/man1/buildif.1man*, even if it already has one.

```
/usr/lib/buildif -f /usr/man/man1/buildif.1man
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

When a file already has an index format table and the **—f** option is given, the old table is removed before the new index is built. If the new index format table can not be built for some reason, the old index will have been removed anyway.

SEE ALSO

apropos(1man), help(1man), makewhatis(1man), man(1man), section(1man), whatis(1man), man(5man), manindex(5man), whatis(5man), catman(8man).

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. The *month* is a number between 1 and 12. *Year* can be between 1 and 9999. The calendar produced is a Julian calendar.

Try September 1752.

EXAMPLES

The following example prints the calendar for 1983:

```
cal 1983
```

This example prints the calendar for the current month:

```
cal `date +%m` 19`date +%y`
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

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.

SEE ALSO

date(1).

NAME

calendar — reminder service

SYNOPSIS

calendar [**—v**] [—]

DESCRIPTION

Calendar consults the file *Calendar* or *calendar* in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. If the public calendar file (*/usr/lib/public.calendar*) exists, **calendar** checks this file in addition to the personal calendar files.

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.

If you give the month as "*" with a date (for example, "* 1") that day in any month will do. Lines that begin with the word *everyday* (or *Everyday*) are considered constant reminders and are always printed by **calendar**.

When the — argument is present, **calendar** mails reminders to all users who have a file **calendar** or **Calendar** in their login directory if either the personal or public calendar file contains entries for today or tomorrow. Normally this is done daily in the wee hours under control of *cron (8).*

The file **calendar** is first run through the C preprocessor, */lib/cpp*, to include any other calendar files specified with the usual *#include* syntax.

OPTIONS

—v Verbose. **Calendar** prints messages telling you what it is doing.

EXAMPLES

This is an example of a calendar file:

```
Feb 14 Valentines Day
12/31  David's birthday
* 1    The first day of the month
Everyday print this constant reminder
```

If **calendar** were executed on December 31 with the above data in the calendar file, the second, third, and fourth entries in the file would be printed.

FILES

Calendar	Alternate name for user's calendar file
calendar	The user's calendar file
/etc/passwd	Used to find logins and home directories for all users
*/tmp/cal**	File for collecting calendar data
/usr/lib/public.calendar	Public calendar

RETURN VALUE

[0] The exit code is always 0.

CAVEATS

Calendar's extended idea of 'tomorrow' doesn't account for holidays.

The program uses the programs */lib/cpp, /usr/lib/calendar, egrep(1), sed(1),* and *mail(1mh)* to do the job, so changes to these may affect **calendar.**

Executing calendar with the dash (—) option will cause mail to be sent to all users with calendar files. It is recommended that this option only be used by **cron.**

SEE ALSO

at(1), egrep(1), mail(1mh), sed(1), cron(8).

NAME

cat — catenate and print

SYNOPSIS

cat [**—b**] [**—e**] [**—n**] [**—s**] [**—t**] [**—u**] [**—v**] [*filename ...*]

DESCRIPTION

Cat reads each *filename* in sequence and displays it on the standard output. Thus

```
cat file
```

displays the file on the standard output, and

```
cat file1 file2 >file3
```

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

If no input file is given, or if the dash argument (—) is encountered, **cat** reads from the standard input. Output is buffered in 1024–byte blocks unless the standard output is a terminal, in which case it is line buffered.

OPTIONS

—b Displays the non–blank output lines preceded by line numbers, sequentially numbered from one.

—e Similar to **—v**, with the exception that it displays a $ character at the end of each line.

—n Displays the output lines preceded by line numbers, sequentially numbered from one.

—s Removes multiple empty lines, leaving only one blank line. Note that a line which contains only blanks and tabs is *not* an empty line.

—t Similar to **—v**, with the exception that it displays tab characters as ^I.

—u Causes the output to be completely unbuffered.

—v Displays non–printing characters so that they are visible. Control characters print like ^X for ⟨CTRL–X⟩ (except for tabs which continue to be expanded); the delete character (octal 0177) prints as ^?. Non–ascii characters (with the high bit set) are printed as M– (for meta) followed by the character of the low 7 bits.

EXAMPLES

The following example concatenates the files *tmp.1, tmp.2* and *tmp.3* onto standard output.

```
cat tmp.1 tmp.2 tmp.3
```

DIAGNOSTICS

cat: Output is going to be written on input file **file**

Cat displays this message if you try to redirect its output to a file which was one of the input files — in other words, **cat** won't overwrite one of its input files. (See note below.)

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

Sh(1sh) redirection such as **cat a b >a** and **cat a b >b**, may destroy the input files before reading them.

SEE ALSO

cp(1), ex(1), more(1), pr(1), sh(1sh), tail(1).

NAME

cb — C program beautifier

SYNOPSIS

cb

DESCRIPTION

Cb places a copy of the C program from the standard input on the standard output with spacing and indentation that displays the structure of the program.

EXAMPLES

The following example formats the file *hello.c* and puts the formatted copy in the file *hellof.c*.

```
cb < hello.c > hellof.c
```

RETURN VALUE

[NO_ERRS] Command completed without error.

SEE ALSO

expand(1), unexpand(1).

NAME

cc — C compiler

SYNOPSIS

cc [**—c**][**—go**][**—o** *output*][**—p**][**—pg**][**—t[p012]**][**—w**] [**—B** *string*][**—C**][**—D***name*=*def*][**—D***name*][**—E**][**—I***dir*] [**—M**][**—O**][**—R**][**—S**][**—U***name*][**—l***library*]*filename...*

DESCRIPTION

Cc is the UTek 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, however, if a single C program is compiled and loaded all at one go.

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

Other arguments are taken to be either loader 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 loaded (in the order given) to produce an executable program with name **a.out.**

OPTIONS

The following options are interpreted by **cc**:

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

—go
Have the compiler produce additional symbol table information for *adb (1)* or *sdb (1)*.

—l*library*
Pass the given *library* to **ld**. The option **—lc** is the last to be passed on to **ld**. This option must be given after the filenames on the command line. See *ld (1)* for more detail about load–time options.

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

—p Arrange for the compiler to produce code which counts the number of times each routine is called. If loading 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. Also, a profiling library is searched, in lieu of the standard C library. An execution profile can then be generated by use of *prof (1)*.

—pg
Causes the compiler to produce counting code in the manner of **—p,**

but invokes a run-time recording mechanism that keeps more extensive statistics and produces a *gmon.out* file at normal termination. An execution profile can then be generated by use of *gprof (1)*.

—t [**p012**]
Find only the designated compiler passes in the files whose names are constructed by a **—B** option. In the absence of a **—B** option, the *string* is taken to be */usr/c/*.

—w
Suppress warning diagnostics.

—B*string*
Find substitute compiler passes in the files named *string* with the suffixes *cpp*, *ccom*, and *c2*. If *string* is empty, use a standard backup version.

—C Prevent the macro preprocessor from eliding comments.

—D*name* = *def*

—D*name*
Define the *name* to the preprocessor, as if by *#define*. If *def* is not given, the name is defined as **1**.

—E Run only the macro preprocessor on the named C programs, and send the result to the standard output.

—M
Run only the macro preprocessor on the named C programs, and have it generate a *make (1)* dependency list.

—I*dir*
#include files whose names do not begin with a slash / are always sought first in the directory of the *filename* argument, then in directories named in **—I** options, then in directories on a standard list.

—O
Invoke an object-code improver.

—R Passed on to **as**, making initialized variables shared and read-only.

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

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

FILES

file.c	input file
file.o	object file
file.a	object code archive file
a.out	loaded output (default)

/tmp/ctm??????	temporary
/lib/cpp	preprocessor
/lib/ccom	compiler
/lib/c2	optional optimizer
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/usr/lib/gcrt0.o	startoff for *gprof(1)*–profiling
/lib/libc.a	standard library, see *intro(3)*
/usr/lib/libc_p.a	profiling library, see *intro(3)*
/usr/include	standard directory for *#include* files
mon.out	file produced for analysis by *prof(1)*
gmon.out	file produced for analysis by *gprof(1)*

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self–explanatory. Messages labelled *cpp* are generated during the preprocessing phase and messages labelled *ccom* are generated during the compilation phase. Occasional messages may be produced by the assembler or loader. These are labelled *as* and *ld* respectively.

SEE ALSO

adb(1), ar(1), as(1). gprof(1), ld(1), prof(1), ranlib(1), make(1), monitor(3c).

NAME

ccat — catenate and print compressed data

SYNOPSIS

ccat [*filename ...*]

DESCRIPTION

Ccat cats data compressed by **compact.** If file arguments are given, the data in the files is uncompressed to the standard output, leaving the files compressed. If no filenames are given, **ccat** reads from standard input, just like *uncompact(1).*

EXAMPLES

This example copies the data in *prog.c.C* (the result of compacting the file *prog.c)* to the standard output:

```
ccat prog.c.C
```

FILES

**.C*	Compacted data file created by *compact(1)*

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	One of the files specified was not found or not readable.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

compact(1), uncompact(1).

NAME

cd, chdir, pushd, popd, dirs — directory change commands (**csh** *built-in)*

SYNOPSIS

cd [*dirname*]
or
cd *+n*

chdir [*dirname*]
or
chdir *+n*

pushd [*dirname*]
or
pushd *+n*

popd [*+n*]

dirs

DESCRIPTION

Csh maintains the name current directory and a directory stack which can be used to keep up with all of the places the user has been and may wish to go back to. With this, the user does not have to remember where to go back to when sidetracked.

The command **cd,** and its synonym **chdir,** change the current working directory of the shell. If no argument is given, the directory is changed to the user's home directory. If *dirname* begins with '/', './', or '../', an attempt is made to change to that directory. Otherwise, the directory *dirname* is searched for in the current directory and in each element of the variable *cdpath.* If this fails, and there is a variable named the same as the value of *dirname* whose value begins with a '/', that directory is used (see EXAMPLES). When the directory is changed, the top element of the directory stack is replaced by the new directory. When the *cdpath* or directory name variable is used to change the directory, the path of the new current directory is printed. With the argument *+n,* where *n* is a number (beginning at 0), the *n*'th element of the directory stack is moved from that position to the top of the stack. The contents of the directory stack is always printed upon completion.

The command **pushd** is used to add directory names to the directory stack, and to edit the stack, as well as change the current directory. With no arguments, **pushd** exchanges the top two elements of the directory stack. With a directory name argument, the directory is changed, and the name of the new current directory is pushed on to the stack. With the argument *+n,* where *n* is a number (beginning at 0), the *n*'th element of the directory stack is moved from that position to the top of the stack. The contents of the directory stack is always printed upon completion.

The command **popd** is used to remove elements from the directory stack. With no arguments, the top element of the stack is removed, resulting in changing the current working directory. With the argument *+n,* where *n* is a number (numbers begin at 0, but '+0' is not valid), the *n*'th element of the directory stack is removed. In the latter use, the current directory is unchanged. The contents of the directory stack is always printed upon completion.

The command **dirs** prints the contents of the directory stack in order. The top element (which is the current directory) is printed first.

The variable *cdpath* contains a list of directories to search if the directory name given is not a subdirectory of the current directory. This variable is maintained along with the environment variable CDPATH, which is used by the *sh(1sh)* **cd** command, but these variables are not the same. In **csh,** the current directory is implicitly the first element in *cdpath,* whereas in **sh,** the current directory must be given explicitly. In order to cope with this difference, the value of CDPATH is imported upon startup of a new shell. CDPATH is changed when *cdpath* is changed (using *set(1csh)),* but *cdpath* is not changed when CDPATH is changed (using *setenv(1csh)).*

The variable *cwd* is set by **csh** whenever the current directory is changed. Due to symbolic links and the ditributed file system, this variable may not always contain correct or desirable data. If the value of *cwd* is required to be correct, the variable *hardpaths* may be set, which causes all changes of directory to get the current directory path by calling *getwd(3c),* which will give the correct path. This makes directory changes somewhat slower. An alternate method is to use the command *pwd(1)* to get the correct current directory path when needed.

EXAMPLES

This example shows a use of the directory name variable.

```
set default=/
set cdpath=( ~ ~/* )
cd default
```

In this case, if there is no directory named 'default' in the current directory, the user's home directory, and any subdirectories of the user's home directory, the current directory will become '/'.

The following example shows some of the features of manipulation of the directory stack. Here, the character '%' represents the **csh** prompt.

```
% dirs
~

% pushd /bin
/bin ~
```

```
% pushd /etc
/etc /bin ~
% cd /usr
% dirs
/usr /bin ~
% pushd +2
~ /usr /bin
% popd +2
~ /usr
% popd
/usr
```

VARIABLES

CDPATH	The directory change search path.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the error message occurred.

CAVEATS

Shell scripts should never change to a subdirectory and attempt to go back by executing **cd ..** , since the directory changed to may be a symbolic link to another directory whose parent directory is different from where the last **cd** was executed. At the very least, the *cwd* variable's value should be saved and used to go back.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1sh), chdir(1sh), continue(1csh), csh(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), chdir(2).

NAME

cd, chdir — change the current directory (*sh* built–in)

SYNOPSIS

cd [*directory*]

DESCRIPTION

Cd changes the current directory to *directory*. The value of the environment variable **HOME** is used if no directory is given.

The environment variable **CDPATH** specifies the directory search path, which is a colon–separated list of directories. The default value of **CDPATH** is the null string, which specifies that only the current directory is to be searched. The current directory may be specified anywhere in the search path by either a null entry or the entry of a dot (.).

If the specified directory contains a slash (/), the directory search path is ignored.

The shell command **chdir** is the same as **cd**.

EXAMPLES

The following example changes the current directory to */bin:*

```
cd /bin
```

This means that references to files in */bin* can be made by the base name of the file rather than */bin/name.*

VARIABLES

CDPATH	Colon–separated list of directories to search for the named directory.
HOME	The default directory to change to.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

In shell scripts, a failed **cd** causes the shell script to be terminated.

In *csh(1csh),* the shell variable **cdpath** is slightly different from **CDPATH.** For more information, see *cd(1csh).*

SEE ALSO

break(1sh), cd(1csh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), chdir(2), execve(2).

NAME

ceil — ceiling function

SYNOPSIS

ceil [**—c***n*] [*vector ...*]

DESCRIPTION

Output is a vector with each element being the smallest integer greater than the corresponding element from the input *vector(s)*. If no *vector* is given, the standard input is assumed.

OPTIONS

—c*n*

n is the number of output elements per line.

EXAMPLES

The following example outputs the ceiling of each element of A, three per line.

```
ceil -c3 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

cflow — generates C flow graph

SYNOPSIS

cflow [**—r**] [**—ix**] [**—i_**] [**—d***num*] *filename . . .*

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), respectively, 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** suffixed 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 number (for example, 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 (for example, **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 filename and location counter under which the symbol appeared (for example, *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.

OPTIONS

—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.

EXAMPLES

As an 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                       i: int, <file.c 1>
5               g: <>
```

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.

DIAGNOSTICS

Complains about bad options.

Complains about multiple definitions and only believes the first.

Other messages may come from the various programs used (for example, from the C–preprocessor).

CAVEATS

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.

SEE ALSO

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

NAME

cd, chdir, pushd, popd, dirs — directory change commands (**csh** *built-in)*

SYNOPSIS

cd [*dirname*]
or
cd *+n*

chdir [*dirname*]
or
chdir *+n*

pushd [*dirname*]
or
pushd *+n*

popd [*+n*]

dirs

DESCRIPTION

Csh maintains the name current directory and a directory stack which can be used to keep up with all of the places the user has been and may wish to go back to. With this, the user does not have to remember where to go back to when sidetracked.

The command **cd,** and its synonym **chdir,** change the current working directory of the shell. If no argument is given, the directory is changed to the user's home directory. If *dirname* begins with '/', './', or '../', an attempt is made to change to that directory. Otherwise, the directory *dirname* is searched for in the current directory and in each element of the variable *cdpath.* If this fails, and there is a variable named the same as the value of *dirname* whose value begins with a '/', that directory is used (see EXAMPLES). When the directory is changed, the top element of the directory stack is replaced by the new directory. When the *cdpath* or directory name variable is used to change the directory, the path of the new current directory is printed. With the argument *+n,* where *n* is a number (beginning at 0), the *n*'th element of the directory stack is moved from that position to the top of the stack. The contents of the directory stack is always printed upon completion.

The command **pushd** is used to add directory names to the directory stack, and to edit the stack, as well as change the current directory. With no arguments, **pushd** exchanges the top two elements of the directory stack. With a directory name argument, the directory is changed, and the name of the new current directory is pushed on to the stack. With the argument *+n,* where *n* is a number (beginning at 0), the *n*'th element of the directory stack is moved from that position to the top of the stack. The contents of the directory stack is always printed upon completion.

The command **popd** is used to remove elements from the directory stack. With no arguments, the top element of the stack is removed, resulting in changing the current working directory. With the argument *+n,* where *n* is a number (numbers begin at 0, but '+0' is not valid), the *n*'th element of the directory stack is removed. In the latter use, the current directory is unchanged. The contents of the directory stack is always printed upon completion.

The command **dirs** prints the contents of the directory stack in order. The top element (which is the current directory) is printed first.

The variable *cdpath* contains a list of directories to search if the directory name given is not a subdirectory of the current directory. This variable is maintained along with the environment variable CDPATH, which is used by the *sh(1sh)* **cd** command, but these variables are not the same. In **csh,** the current directory is implicitly the first element in *cdpath,* whereas in **sh,** the current directory must be given explicitly. In order to cope with this difference, the value of CDPATH is imported upon startup of a new shell. CDPATH is changed when *cdpath* is changed (using *set(1csh)),* but *cdpath* is not changed when CDPATH is changed (using *setenv(1csh)).*

The variable *cwd* is set by **csh** whenever the current directory is changed. Due to symbolic links and the ditributed file system, this variable may not always contain correct or desirable data. If the value of *cwd* is required to be correct, the variable *hardpaths* may be set, which causes all changes of directory to get the current directory path by calling *getwd(3c),* which will give the correct path. This makes directory changes somewhat slower. An alternate method is to use the command *pwd(1)* to get the correct current directory path when needed.

EXAMPLES

This example shows a use of the directory name variable.

```
set default=/
set cdpath=( ~ ~/* )
cd default
```

In this case, if there is no directory named 'default' in the current directory, the user's home directory, and any subdirectories of the user's home directory, the current directory will become '/'.

The following example shows some of the features of manipulation of the directory stack. Here, the character '%' represents the **csh** prompt.

```
% dirs
~

% pushd /bin
/bin ~
```

```
% pushd /etc
/etc /bin ~
% cd /usr
% dirs
/usr /bin ~
% pushd +2
~ /usr /bin
% popd +2
~ /usr
% popd
/usr
```

VARIABLES

CDPATH	The directory change search path.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the error message occurred.

CAVEATS

Shell scripts should never change to a subdirectory and attempt to go back by executing **cd .. ,** since the directory changed to may be a symbolic link to another directory whose parent directory is different from where the last **cd** was executed. At the very least, the *cwd* variable's value should be saved and used to go back.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1sh), chdir(1sh), continue(1csh), csh(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), chdir(2).

NAME

cd, chdir — change the current directory (*sh* built-in)

SYNOPSIS

cd [*directory*]

DESCRIPTION

Cd changes the current directory to *directory*. The value of the environment variable **HOME** is used if no directory is given.

The environment variable **CDPATH** specifies the directory search path, which is a colon-separated list of directories. The default value of **CDPATH** is the null string, which specifies that only the current directory is to be searched. The current directory may be specified anywhere in the search path by either a null entry or the entry of a dot (.).

If the specified directory contains a slash (/), the directory search path is ignored.

The shell command **chdir** is the same as **cd**.

EXAMPLES

The following example changes the current directory to */bin:*

```
cd /bin
```

This means that references to files in */bin* can be made by the base name of the file rather than */bin/name.*

VARIABLES

CDPATH	Colon-separated list of directories to search for the named directory.
HOME	The default directory to change to.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

In shell scripts, a failed **cd** causes the shell script to be terminated.

In *csh(1csh),* the shell variable **cdpath** is slightly different from **CDPATH.** For more information, see *cd(1csh).*

SEE ALSO

break(1sh), cd(1csh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), chdir(2), execve(2).

NAME

checkeq — check neqn files for syntax

SYNOPSIS

checkeq [*filename* . . .]

DESCRIPTION

Checkeq reports missing or unbalanced delimiters and **.EQ/.EN** pairs in input for *neqn(1)*. If no filenames are given, the input is read from the standard input.

EXAMPLES

This example checks the file *example.eq* for possible errors:

```
checkeq example.eq
```

RETURN VALUE

[NO_ERRS] Command completed without error.

[NP_WARN]
An error was found in the input.

SEE ALSO

checknr(1), neqn(1), nroff(1), tbl(1), eqnchar(7).

NAME

checknr — check nroff files

SYNOPSIS

checknr [**—a**.*x1.y1.x2.y2. ... .xn.yn*] [**—c**.*x1.x2.x3 ... .xn*] [**—f**] [**—s**] [*filename ...*]

DESCRIPTION

Checknr checks a list of *nroff (1)* input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. If no files are specified, **checknr** checks the standard input. Delimiters checked are:

(1) Font changes using `\fx ... \fP`.

(2) Size changes using `\sx ... \s0`.

(3) Macros that come in open ... close forms, for example, the **.TS** and **.TE** macros, which must always come in pairs.

Checknr knows about the *ms (7)* and *me (7)* macro packages.

Checknr is intended to be used on documents that are prepared with **checknr** in mind, much the same as **lint.** It expects a certain document writing style for **\f** and **\s** commands, in that each **\f***x* must be terminated with **\fP** and each **\s***x* must be terminated with **\s0**. While it will work to directly go into the next font or explicitly specify the original font or point size, and many existing documents actually do this, such a practice will produce complaints from **checknr.** Since it is probably better to use the **\fP** and **\s0** forms anyway, you should think of this as a contribution to your document preparation style.

OPTIONS

—a Adds additional pairs of macros to the list of macros checked. The —a option must be followed by groups of six characters, each group defining a pair of macros. The six characters are a period, the first macro name, another period, and the second macro name. For example, to define a pair **.BS** and **.ES**, use **—a.BS.ES**

—c Defines commands which would otherwise be complained about as undefined.

—f Requests **checknr** to ignore **\f** font changes.

—s Requests **checknr** to ignore **\s** size changes.

DIAGNOSTICS

Complaints about unmatched delimiters.
Complaints about unrecognized commands.
Various complaints about the syntax of commands.

CAVEATS

There is no way to define a one–character macro name using **—a.**

Files from different documents should be processed separately because macro definitions are kept for the entire execution.

SEE ALSO

nroff(1), me(7), ms(7).

NAME

chfn — change full name entry in password file

SYNOPSIS

chfn [*loginname*]

DESCRIPTION

Chfn is used to change password file information about users. This information is used by query programs, such as **finger**. It consists of information such as the user's "real life" name, office room number, office phone number, and home phone number. **Chfn** prompts the user for each field. Included in the prompt is a current value, which is enclosed between brackets. The current value is retained simply by pressing ⟨RETURN⟩. To enter a blank field, type the word *none*.

The optional argument *loginname* is normally used by the superuser to change another person's **finger** information.

Synchronization is enforced to prevent simultaneous multiple-user updates of the *passwd* file.

EXAMPLES

Below is sample output from a call to **chfn:**

```
Name [Biff Studsworth II]:
Room number (Exs: 597E or 197C) []:  521E
Office Phone (Ex: 1632) []:  1863
Home Phone (Ex: 987532) [5771546]:  none
```

FILES

/etc/passwd	System user information file
/etc/ptmp	Password lock file.
/usr/lib/chfn	Finger information format.

CAVEATS

It is a good idea to run **finger** after running **chfn** to make sure everything is the way you want it.

The prompts are from a site-dependent *prompt* file, see *chfn(5)*.

For historical reasons, the user's name, and so forth are stored in the *passwd* file. This is a bad place to store the information. A data base is being developed to store this information.

SEE ALSO

ed(1), finger(1n), passwd(5), regex(3c).

NAME

chgrp — change group ownership

SYNOPSIS

chgrp [**—f**] [**—l**] *group filename...*

DESCRIPTION

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

The user invoking **chgrp** must belong to the specified group, or be the superuser. OPTIONS

—f Force. No error messages about nonexistent files or files that can not have group ownership changed.

—l Follow symbolic links. Normally, the group ownership of the symbolic link itself is changed. The **—l** option causes **chgrp** to follow the symbolic links and change the group ownership of the file pointed to.

EXAMPLES

The following invocation changes the group of the file *temp* to *system*:

```
chgrp system temp
```

FILES

/etc/group	System group information

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

chmod(1), ln(1), chgrp(2), chown(2), passwd(5), group(5), chown(8).

NAME

chmod — change mode of file or directory

SYNOPSIS

chmod *mode filename . . .*

DESCRIPTION

The mode of each named file is 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 all, or **ugo.** If *who* is omitted, the default is **a** but the setting of the file creation mask (see *umask(2)*) is taken into account.

Op can be + to add *permission* to the file's mode, — (a minus sign) to take away *permission*, and = (an equal sign) 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 — sticky). Letters **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.**

Only the owner of a file (or the superuser) may change its mode.

EXAMPLES

The first example denies write permission to others; the second makes a file executable:

```
chmod o-w file
chmod +x file
```

This example uses the permissions that are on for the group and turns them on for the user and others:

```
chmod uo+g
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

ls(1), chmod(2), stat(2), umask(2), chown(8).

NAME

chsh — change default login shell

SYNOPSIS

chsh *name* [*shell*]

DESCRIPTION

Chsh is a command similar to *passwd (1)* except that it is used to change the login shell field of the password file rather than the password entry. If no *shell* is specified then the shell reverts to the default login shell */bin/sh*. Otherwise, any executable file can be specified as the shell. If the full path to a file is not specified, for example, the name does not start with "*/*", **chsh** will prepend */bin*.

EXAMPLES

An example use of this command would be:

```
chsh bill /bin/csh
```

CAVEATS

If the file exists and is executable, it will be installed as your login shell. Make sure you really want that file as your shell. An error may make login impossible and super–user intervention necessary.

SEE ALSO

chfn(1), csh(1csh), passwd(1), passwd(5).

NAME

ci — check in RCS revisions

SYNOPSIS

ci [—**N***name*] [—**P**] [—**d***[rev]*] [—**f***[rev]*] [—**k***[rev]*] [—**l***[rev]*] [—**m***msg*] [—**n***name*] [—**q***[rev]*] [—**r***[rev]*] [—**s***state*] [—**t***txtfilename*] [—**u***[rev]*] *filename ...*

DESCRIPTION

Ci stores new revisions into RCS files. Each filename ending in *,v* is considered an RCS file; all others are assumed to be working files containing new revisions. **Ci** deposits the contents of each working file into the corresponding RCS file.

Pairs of RCS files and working files may be specified in three ways (see also the EXAMPLES section of *co(1rcs)*):

1) Both the RCS file and the working file are given. The RCS filename is of the form *pathname1/workfilename,v* and the working filename is of the form *pathname2/workfilename*, where *pathname1/* and *pathname2/* are (possibly different or empty) paths and *workfilename* is a filename.

2) Only the RCS file is given. Then the working file is assumed to be in the current directory and its name is derived from the name of the RCS file by removing *pathname1/* and the suffix *,v*.

3) Only the working file is given. Then the name of the RCS file is derived from the name of the working file by removing *pathname2/* and appending the suffix *,v*.

If the RCS file is omitted or specified without a path, then **ci** looks for the RCS file first in the directory *./RCS* and then in the current directory.

For **ci** to work, your login name must be on the access list, unless the access list is empty or you are the superuser or the owner of the file. To append a new revision to an existing branch, the tip revision on that branch must be locked by the caller. Otherwise, only a new branch can be created. This restriction is not enforced for the owner of the file, unless locking is set to *strict* (see *rcs(1rcs)*). A lock held by someone else may be broken with the **rcs** command and its options.

If the revision to be deposited is not different from the preceding one, **ci** either aborts the deposit (if **–q** is given) or asks whether to abort (if **–q** is omitted). The current revision is unlocked in this case, unless the **—d** or **—l** flag is given. If the **—u** flag is given, the file is unlocked, removed, and checked out again. A deposit can be forced with the **–f** option.

For each revision deposited, **ci** prompts for a log message that summarizes the change. You can enter the text editor to edit the log message by pressing ⟨ESC⟩. The environment variables RCSEDIT, EDITOR, and EDIT are checked in order to determine the text editor to invoke. If none of these are set, *vi(1)* is invoked. When you finish entering the log message, enter a line containing only a single dot (.) or a ⟨CTRL–D⟩.

If several files are checked in, **ci** asks whether to reuse the previous log message. If the *standard* input is not a terminal, **ci** suppresses the prompt and uses the same log message for all files. See also **–m**.

You can specify the number of the deposited revision with one of the options: **–r**, **–d**, **–f**, **–k**, **–l**, **–u**, or **–q** (see **–r**).

If the RCS file does not exist, **ci** creates it and deposits the contents of the working file as the initial revision (default number: 1.1). The access list is initialized to empty. Instead of the log message, **ci** requests descriptive text (see **–t** below).

An RCS file created by **ci** inherits the read and execute permissions from the working file. If the RCS file exists already, **ci** preserves its read and execute permissions. **Ci** always turns off all write permissions of RCS files.

OPTIONS

—d[*rev*]
If the revision has not changed and the checkin is aborted, this option causes the current revision to remain locked.

—f[*rev*]
Force a deposit. The new revision is deposited even it is not different from the preceding one.

—k[*rev*]
Searches the working file for keyword values to determine its revision number, creation date, author, and state (see *co(1rcs)*), and assigns these values to the deposited revision, rather than computing them.

This option is often used for software distribution. A revision that is sent to several sites should be checked in with the **–k** option at these sites to preserve its original number, date, author, and state.

—l[*rev*]
Works like **–r**, except it performs an additional **co –l** for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is often used when you want to save a revision and continue editing it after the checkin.

—m*msg*
Use *msg* as the log message for all revisions checked in.

—n*name*
Assign the symbolic name *name* to the checked–in revision. **Ci** prints an error message if *name* is already assigned to another number.

—q[*rev*]
Quiet mode. Diagnostic output is not printed. A revision that is the same as the preceding one is not deposited, unless **–f** is given.

—r[*rev*]
Assigns the revision number *rev* to the checked–in revision, releases the corresponding lock, and deletes the working file. This is the default.

If *rev* is omitted, **ci** derives the new revision number from your last lock. If you locked the tip revision of a branch, the new revision is appended to that branch. The new revision number is obtained by incrementing the tip revision number. If you locked a nontip revision, a new branch is started at that revision by incrementing the highest branch number at that revision. The default initial branch and level numbers are 1. If you don't hold a lock, but you are the owner of the file and locking is not set to *strict*, then the revision is appended to the trunk.

If *rev* indicates a revision number, it must be higher than the latest one on the branch to which *rev* belongs, or must start a new branch.

If *rev* indicates a branch instead of a revision, the new revision is appended to that branch. The level number is obtained by incrementing the tip revision number of that branch. If *rev* indicates a nonexisting branch, that branch is created with the initial revision numbered *rev.1*.

Exception: On the trunk, revisions can be appended to the end, but not inserted.

—s*state*
Set the state of the checked–in revision to *state*. The default is *Exp*.

—t [*txtfilename*]
Write descriptive text into the RCS file (deleting the existing text). If *txtfilename* is omitted, **ci** prompts you for text supplied from the *standard* input, terminated with a line containing a single dot (.) or ⟨CTRL–D⟩. Otherwise, the descriptive text is copied from *txtfilename*. During initialization, descriptive text is requested even if **–t** is not given. The prompt is suppressed if *standard* input is not a terminal.

—u [*rev*]
Works like **–l**, except that the deposited revision is not locked. This is often used when you want to use the revision immediately after checkin.

—P
The access and modification dates of the RCS file (and the working file if the **—l** or **—u** option is given) are made the same as the original working file. This option is supplied for special applications involving very large projects with many file dependencies but should generally not be used because of the problems that can occur. See CAVEATS.

—N*name*
Same as **–n**, except that it overrides a previous assignment of *name*.

EXAMPLES

The following example checks the file *example.c* into the RCS file */usr/lib/RCS/example.c,v* and leaves a copy of the file in the current directory. The file in the current directory will not be writeable:

```
ci -u example.c /usr/lib/RCS/example.c,v
```

If the current directory is */usr/lib* or */usr/lib/RCS*, the last argument may be ommitted without changing the result of the command.

FILES

,RCSt$$	Temporary storage for checking in the revision.
,,*	Semaphore file. This file prevents modifications to the working file by other RCS programs while being checked in.

DIAGNOSTICS

For each revision, **ci** prints the RCS file, the working file, and the number of both the deposited and the preceding revision.

VARIABLES

RCSLOCK	If set to *nonstrict*, all new RCS files will have locking set to non–strict mode. Otherwise, locking is strict.
RCSEDIT	Checked for the name of a text editor to invoke.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

You must have read and write permission for the directories containing the RCS file and the working file, and read permission for the RCS file itself.

A number of temporary files are created. A semaphore file is created in the directory containing the RCS file. **Ci** always creates a new RCS file and unlinks the old one. This strategy makes links to RCS files useless.

The maximum length of a log message is 2048 characters. The maximum length of a description is 2048. Longer messages are truncated.

The maximum number of revisions that can be stored in a single RCS file is 719. When there are more than 700 revisions in a file, a warning message is printed on the terminal (if possible) every time an RCS command works on the file. See *rcsfile(5rcs)* for information on what action to take in this case. **Ci** does not allow the 720th revision to be checked in.

On older versions of RCS, the maximum number of revisions that can be stored in a single RCS file is 239. No warning message is displayed on the terminal if this number is exceeded.

The **—P** option is supplied so that **make** does not think that a file has been changed just because it was checked in (see *make(1)*). This causes **make** to do less work, but it can cause problems if the values of any of the RCS keyword values are used. For example, if a revision logging system is used to store the version numbers of source files into the object code, these numbers may not be correct if the **—P** option is used. It is best not to use this option, which is supplied for people building very large projects.

Unless the **—I** option is used, the *Locker* keyword is not expanded.

SEE ALSO

co(1rcs), ident(1rcs), rlog(1rcs), rcs(1rcs), rcsdiff(1rcs), rcsintro(1rcs), rcsmerge(1rcs), rcsfile(5rcs).

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 and then in the termcap entry for that terminal to figure out how to clear the screen.

EXAMPLES

The following example clears the terminal screen:

```
clear
```

FILES

/etc/termcap	Default terminal capability data base

VARIABLES

TERM	The type of terminal being used.
TERMCAP	The name of the terminal capability entry, or the actual entry.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	No TERM environment variable.
[2]	No termcap entry for the specified terminal type.
[3]	No clear-screen sequence in the termcap entry.

SEE ALSO

termcap(5t).

NAME

cmp — compare two files

SYNOPSIS

cmp [**—l**] [**—s**] *filename1 filename2*

DESCRIPTION

The two files are compared. (If *filename1* is a dash (—), 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.

EXAMPLES

The following compares a local version of echo with the version in */bin:*

```
cmp echo /bin/echo
```

RETURN VALUE

[NO_ERRS]
Files are identical.

[1]	Files are different
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

If both option flags are used they must be in the order shown. Use of both is identical to a call with no options.

SEE ALSO

diff(1), comm(1), sdiff(1).

NAME

co — check out RCS revisions

SYNOPSIS

co [—**P**] [—**d***date*] [—**j***joinlist*] [—**l***[rev]*] [—**p***[rev]*]
[—**q***[rev]*] [—**r***[rev]*] [—**s***state*] [—**u***login*] [—**w***[login]*] *filename ...*

DESCRIPTION

Co retrieves revisions from RCS files. Each filename ending in *,v* is taken to be an RCS file. All other files are assumed to be working files. **Co** retrieves a revision from each RCS file and stores it into the corresponding working file.

You may specify pairs of RCS files and working files in three ways:

1) Give both the RCS filename and the working filename. The format of an RCS filename is: *pathname1/workfilename,v*. The format of a working filename is: *pathname2/workfilename*. *Pathname1/* and *pathname2/* are (possibly different or empty) paths and *workfilename* is a filename.

2) Give only the RCS filename. The working file is created in the current directory and its name is derived from the name of the RCS file by removing *pathname1/* and the suffix *,v*.

3) Give only the working filename. The name of the RCS file is derived from the name of the working file by removing *pathname2/* and appending the suffix *,v*.

If you omit the RCS filename or specify it without a path, **co** looks for the RCS file in the directory *./RCS* and then in the current directory.

You may check out revisions of an RCS file locked or unlocked. Locking a revision prevents overlapping updates. You need not lock a revision that is checked out for reading or processing (for example, compiling). A revision you checked out for editing and later checkin should be locked. **Co** won't let you lock a revision that is currently locked by another user. (A lock may be broken with the *rcs(1rcs)* command.)

To check out a file locked, you must be:

1) On the access list of the RCS file, or
2) The owner of the file, or
3) Logged in as the superuser.

You can also checkout a file locked if the access list is empty. **Co** without locking is not subject to access-list restrictions.

You may select a revision by number, checkin date/time, author, or state. When these options are applied in combination, the latest revision that satisfies all of them is retrieved. If you don't specify any of these options **co** retrieves the latest revision on the trunk.

The options for date/time, author, and state retrieve a revision on the *selected branch*. The selected branch is either derived from the revision number (if given), or is the highest branch on the trunk. You may specify a revision number with one of the options: **-l**, **-p**, **-q**, or **-r**.

A **co** command applied to an RCS file with no revisions creates a zero-length file. **Co** always performs keyword substitution (see below).

Co creates the working file with the same read and execute permissions as the RCS file. In addition, write permission for the owner of the file is turned on, unless the file is checked out unlocked and locking is set to *strict* (see *rcs(1rcs)*).

If a file with the name of the working file exists already and you have write permission, **co**:

1) Aborts the checkout if **-q** is given, or
2) Asks whether to abort if **-q** is not given.

If the working file exists but is not writable, it is deleted before the checkout.

KEYWORD SUBSTITUTION

Strings of the form *$keyword$* and *$keyword:...$* embedded in the text are replaced with strings of the form *$keyword: value $*, where *keyword* and *value* are pairs listed below. (Note: the Locker keyword is only expanded in the working file that was actually checked out with the lock.) Keywords may be embedded in literal strings or comments to identify a revision.

Initially, you enter strings of the form *$keyword$*. On checkout, **co** replaces these strings with strings of the form *$keyword: value $*. If a revision containing strings of the latter form is checked back in, the value fields are replaced during the next checkout. Thus, the keyword values are automatically updated on checkout.

Keywords and their corresponding values are:

$*Author*$
The loginname of the user who checked in the revision.

$*Date*$
The date and time the revision was checked in.

$*Header*$
A standard header containing the RCS filename, the revision number, the date, the author, and the state.

$*Locker*$
The loginname of the user who locked the revision (empty if not locked).

$*Log*$
The log message supplied during checkin, preceded by a header containing the RCS filename, the revision number, the author, and the date. Existing log messages are *not* replaced. Instead, the new log message is inserted after *$Log:...$*. This is useful for accumulating a complete change log in a source file.

$*Revision*$
The revision number assigned to the revision.

$*Source*$
The full pathname of the RCS file.

$*State*$
The state assigned to the revision with **rcs –s** or **ci –s**.

OPTIONS

—P
Causes the access and modification dates of the working file and RCS to be the same as the original RCS file. See CAVEATS.

—d*date*
Retrieves the latest revision on the selected branch whose checkin date/time is less than or equal to *date*. The date and time may be given in free format and are converted to local time. Examples of formats for *date* are:

```
22-April-1982, 17:20-CDT,
2:25 AM, Dec. 29, 1983,
Tue-PDT, 1981, 4pm Jul 21 (free format),
Fri, April 16 15:52:25 EST 1982 (output of ctime).
```

Most fields in the date and time may be omitted. **Co** determines the defaults in this order: year, month, day, hour, minute, and second (most to least significant). At least one of these fields must be provided. For omitted fields that are of higher significance than the highest provided field, the current values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date 20, 10:30 defaults to 10:30:00 of the 20th of the current month and current year. The date/time must be quoted if it contains spaces.

—j*joinlist*

Generates a new revision which is the join of the revisions on *joinlist*. *Joinlist* is a comma-separated list of pairs of the form *rev2:rev3*, where *rev2* and *rev3* are (symbolic or numeric) revision numbers.

For the initial such pair, *rev1* denotes the revision selected by the options **–l**, ..., **–r**. For all other pairs, *rev1* denotes the revision generated by the previous pair. (Thus, the output of one join becomes the input to the next.)

For each pair, **co** joins revisions *rev1* and *rev3* with respect to *rev2*. This means that all changes that transform *rev2* into *rev1* are applied to a copy of *rev3*. This is particularly useful if *rev1* and *rev3* are the ends of two branches that have *rev2* as a common ancestor. If *rev1* < *rev2* < *rev3* are on the same branch, joining them generates a new revision which is like *rev3*, but with all changes that lead from *rev1* to *rev2* undone. If changes from *rev2* to *rev1* overlap with changes from *rev2* to *rev3*, **co** prints a warning and includes the overlapping sections, delimited by the lines <<<<<<< *rev1*, = = = = = = =, and >>>>>>> *rev3*.

For the initial pair you may omit *rev2* and the following colon, in which case the common ancestor of *rev1* and *rev3* is used for *rev2*. If you omit *rev2* but leave the following colon, *rev1* is used for *rev2* as well as for *rev1*.

If any of the arguments indicate branches, the latest revisions on those branches are assumed. If the option **–l** is present, the initial *rev1* is locked.

—l [*rev*]

Locks the checked out revision for the caller. If **—l** is omitted, the checked out revision is not locked. See option **–r** for handling of the revision number *rev*.

—p [*rev*]

Prints the retrieved revision on the *standard output* rather than storing it in the working file. This option is useful when **co** is part of a pipe (¦) or when you want the output displayed on your terminal.

—q [*rev*]

Quiet mode. Diagnostics are not printed.

—r [*rev*]

Retrieves the latest revision whose number is less than or equal to *rev*. If *rev* indicates a branch rather than a revision, the latest revision on that branch is retrieved. *Rev* is composed of one or more numeric or symbolic fields separated by a dot (.). The numeric equivalent of a symbolic field is specified with the **–n** option of the commands **ci** and **rcs**.

—**s***state*
Retrieves the latest revision on the selected branch whose state is *state.*

—**u***login*
When locking a revision, use *login* as the locker instead of the invoker's login name. This allows creating separate working branches without requiring all lockers to be present to check out the files.

—**w** [*login*]
Retrieves the latest revision on the selected branch that was checked in by the user with loginname *login*. If *login* is omitted, the caller's login is assumed.

EXAMPLES

Suppose the current directory contains a subdirectory *RCS* with an RCS file *io.c,v*. Then all of the following commands retrieve the latest revision from *RCS/io.c,v* and store it into *io.c*:

```
co io.c; co RCS/io.c,v; co io.c,v;
co io.c RCS/io.c,v; co io.c io.c,v;
co RCS/io.c,v io.c; co io.c,v io.c;
```

FILES

,,*	Lock file; exists during checkout. Prevents others from working on the same file.
,RCSt$$	Temporary file for storing working file data during retrieval. $$ is the process id.

DIAGNOSTICS

The RCS filename, the working filename, and the revision number retrieved are written to the diagnostic output.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

To checkout a revision, you must have write permission in the working directory, read permission for the RCS file, and either read permission (for reading) or read/write permission (for locking) in the directory which contains the RCS file.

The option **–j** does not work for files that contain lines with a single dot (.).

The option **-d** gets confused in some circumstances, and accepts no date earlier than 1970. There is no way to suppress the expansion of keywords, except by writing them differently. In *nroff(1),* this is done by embedding the null-character into the keyword.

In order to interface correctly with *make(1),* the working file is always newer than the RCS file after a checkout.

The maximum number of revisions that can be stored in a single RCS file is 719. When there are more than 700 revisions in a file, a warning message is printed on the terminal (if possible) every time an RCS command works on the file. See the manual page for *rcsfile(5rcs)* for information on what action to take in this case.

On older versions of RCS, the maximum number of revisions that can be stored in a single RCS file is 239. No warning message is displayed on the terminal if that number is exceeded.

The **—P** option is supplied so that **make** does not think that a file has been changed just because it was checked out. This causes **make** to do less work, but it can cause problems if the values of any of the RCS keyword values are used. For example, if a revision logging system is used to store the version numbers of source files into the object code, these numbers may not be correct if the **—P** option is used.

Unless the **—l** option is given, the *Locker* keyword is not expanded.

SEE ALSO

ci(1rcs), ident(1rcs), make(1), rlog(1rcs), rcs(1rcs), rcsdiff(1rcs), rcsintro(1rcs), rcsmerge(1rcs), rcsfile(5rcs).

NAME

col — filter reverse linefeeds

SYNOPSIS

col [**— b**] [**— f**] [**— h**]

DESCRIPTION

Col reads the standard input and writes the standard output. It performs the line overlays implied by reverse line feeds (ESC–7 in ASCII) and by forward and reverse half–linefeeds (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.

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–linefeeds (ESC–9), but will still never contain either kind of reverse line motion.

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

The control characters **SO** (ASCII code 017), and **SI** (016) are assumed to start and end text in an alternate character set. The character set (primary or alternate) associated with each printing character read is remembered; on output, **SO** and **SI** characters are generated where necessary to maintain the correct treatment of each character.

Col normally converts white space to spaces. If the **—h** option is given, white space is converted to tabs.

All control characters are removed from the input except space, backspace, tab, return, newline, ESC (033) followed by 7, 8, 9, SI, SO, or VT (013). This last character is an alternate form of full reverse linefeed, for compatibility with some other hardware conventions. All other nonprinting characters are ignored.

OPTIONS

—b **Col** assumes that the output device in use is not capable of backspacing. In this case, if several characters are to appear in the same place, only the last one read will be taken.

—f The output from **col** may contain forward half–linefeeds (ESC–9), but will still never contain either kind of reverse line motion.

—h White space is converted to tabs.

EXAMPLES

The following runs *tbl(1)* and *nroff(1)* on the file *examp.tbl*, pipes this output through **col**, converting the white space to tabs, and finally piping the output through *more(1)*:

```
tbl examp.tbl ¦ nroff ¦ col -h ¦ more
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

Can't back up more than 128 lines.
No more than 2048 characters, including backspaces, on a line.

SEE ALSO

more(1), nroff(1), tbl(1).

NAME

comb — combine SCCS deltas

SYNOPSIS

comb [—**c***list*][—**o**][—**p***sid*][—**s**] *filename* . . .

DESCRIPTION

Comb generates a shell procedure (see *sh (1sh))* 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. If a name of — (a dash) 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.

The generated shell procedure is written on the standard output.

OPTIONS

Each option 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.

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

—**c***list*
A *list* (see *get (1sccs)* 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.

—**p***SID*
The SCCS identification string (*SID*) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

—**s** This argument causes **comb** to generate a shell procedure which, when run, will produce a report that gives (for each file): the filename, 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.

FILES

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

DIAGNOSTICS

Use *sccshelp (1sccs)* for explanations.

CAVEATS

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.

Non–SCCS files and unreadable files are silently ignored.

SEE ALSO

admin(1sccs), delta(1sccs), get(1sccs), sccshelp(1sccs), prs(1sccs), sccsfile(5sccs).

NAME

comm — select or reject lines common to two sorted files

SYNOPSIS

comm [**—[123]**] *filename1 filename2*

DESCRIPTION

Comm reads *filename1* and *filename2,* which should be ordered in ASCII collating sequence, and produces a three column output: lines only in *filename1;* lines only in *filename2;* and lines in both files. The filename — (a dash) means the standard input.

OPTIONS

—1, **—2**, and/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.

EXAMPLES

The following example will print on standard output only the lines common to *filename1* and *filename2*:

```
comm -12 filename1 filename2
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Multiple options may not be split up. For example, executing the command

```
comm -1 -2 filename1 filename2
```

tells **comm** to print lines only in the file *—2* and common to the files *—2* and *filename1.*

SEE ALSO

cmp(1), diff(1), cut(1), egrep(1), fgrep(1), grep(1), join(1), look(1), paste(1), sort(1), uniq(1).

NAME

comp — compose a message

SYNOPSIS

comp [*filename*][**—editor** *editor*][**—form** *formfilename*][**—use**] [**—nouse**][[**—wait**][[**—nowait**][[**—output**][[**—nooutput**] [[**—nouse**][**—build**]][**—help**]

DESCRIPTION

Comp is used to create a new message to be mailed. If *filename* is not specified, the file named *draft* in the user's MH directory will be used. **Comp** copies a message form to the file being composed and then invokes an editor on the file. The default editor is */bin/prompter*, which may be overridden with the **—editor** switch or with a profile entry **Editor:** or with the environment variable **EDITOR**. (See *prompter(1mh)* for details.) The default message form contains the following elements:

```
To:
Cc:
Subject:
----------
```

If the file named *components* exists in the user's MH directory, it will be used instead of this form. If **—form** *formfilename* is specified, the specified *formfilename* (from the MH directory) will be used as the skeleton. The line of dashes or a blank line must be left between the header and the body of the message for the message to be identified properly when it is sent (see *send(1mh)*). The switch **—use** directs **comp** to continue editing an already started message. That is, if a **comp** (or **repl**, or **forw**) is terminated without sending the message, the message can be edited again via **comp —use** .

If the specified file (or draft) already exists, **comp** will ask if you want to delete it before continuing. A reply of **No** will abort the **comp**, **yes** will replace the existing draft with a blank skeleton, **list** will display the draft, and **use** will use it for further composition.

Upon exiting from the editor, **comp** will ask *What now?* . The valid responses are **list**, to list the draft on the terminal; **format**, to run pretty–up the message (see *fmt(1mh)*); **quit**, to terminate the session and preserve the draft; **quit delete**, to terminate, then delete the draft; **send**, to send the message; **send verbose**, to cause the delivery process to be monitored; **edit**〈 *editor* 〉, to invoke 〈*editor*〉 for further editing; and **edit**, to re–edit using the same editor that was used on the preceding round unless a profile entry 〈**lasteditor**〉**—next:** 〈*editor*〉 names an alternative editor.

Your *.mh_profile* can contain the following entries:

```
Path: To determine the user's MH directory
Editor: To override the use of /bin/prompter
```

```
as the default editor
⟨lasteditor⟩-next: To name an editor to be used after exit
from ⟨lasteditor⟩
```

Comp has the following defaults:

filename defaults to draft
—editor defaults to */bin/prompter*
—nouse
—wait
—output

Comp does not affect either the current folder or the current message.

OPTIONS

—editor *editor*
Overrides the default editor with *editor*.

—form *formfilename*
The specified *formfilename* (from the MH directory) will be used as the skeleton format.

—help
Displays a synopsis of the **comp** command.

—nouse
Causes **comp** to ignore any previously started messages.

—use
Causes **comp** to continue editing an already started message.

—wait
Causes **comp** to wait until the message is sent before terminating.

—nowait
Causes **comp** to terminate immediately after selecting send. This is useful for sending message quickly.

—output
Causes **comp** to print error messages to the terminal..

—nooutput
Causes **comp** to not print error messages.

—build
Causes the message to be built but not sent.

FILES

/usr/lib/mh/components	The message skeleton
⟨mh-dir⟩/components	Used instead of the standard skeleton
$HOME/.mh_profile	The user profile
⟨mh-dir⟩/draft	The default message file
/bin/send	To send the composed message

SEE ALSO

fmt(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), mh(1mh), mhl(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh), mh(5mh), mh_profile(5mh).

NAME

compact — compress files

SYNOPSIS

compact [*filename* ...]

DESCRIPTION

Compact compresses the named files using an adaptive Huffman code. If no filenames are given, the standard input is compacted to the standard output. **Compact** operates as an on–line algorithm. Each time a byte is read, it is encoded immediately according to the current prefix code. This code is an optimal Huffman code for the set of frequencies seen so far. It is unnecessary to prepend a decoding tree to the compressed file since the encoder and the decoder start in the same state and stay synchronized. Furthermore, **compact** and **uncompact** can operate as filters. In particular,

```
... ¦ compact ¦ uncompact ¦ ...
```

operates as a (very slow) no–op.

When an argument *filename* is given, it is compacted and the resulting file is placed in *filename.C*; *filename* is unlinked. The first two bytes of the compacted file code the fact that the file is compacted. This code is used to prohibit recompaction.

The amount of compression to be expected depends on the type of file being compressed. Typical values of compression are: Text (38%), Pascal Source (43%), C Source (36%) and Binary (19%). These values are the percentages of file bytes reduced.

EXAMPLES

The following example compacts the file *prog.c*:

```
compact prog.c
```

The resulting compacted data is in the file *prog.c.C*. The file *prog.c* is removed.

FILES

**.C*	Compacted file created by compact.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

ccat(1), uncompact(1).

NAME

continue — continue execution of loop (**csh** built–in)

SYNOPSIS

continue

DESCRIPTION

The **continue** command is used to continue the execution of the nearest enclosing **while** or **foreach** statement. All command lines up to the corresponding **end** are ignored.

Like *break(1csh)*, **continue** causes the rest of the commands on the current line to be executed, so multiple **continue** commands on the same line will continue execution at the corresponding loop.

EXAMPLES

This example shows a shell script which reads lines from the standard input up to the line "end". If a line begins with the character '#', the word "comment" is printed.

```
#!/bin/csh -f
set x="$<"
while ("$x" != "end")
        if ("$x" =~ \#*) then
                set x="comment"
                continue
        endif
        echo "$x"
        set x="$<"
end
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the message occurred.

CAVEATS

Unlike *continue(1sh)*, the **csh** version of **continue** does not take an argument.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), break(1sh), cd(1csh), chdir(1csh), continue(1sh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

cor — ordinary correlation coefficient

SYNOPSIS

cor [**—F***vector*] [*vector ...*]

DESCRIPTION

Output is the ordinary correlation coefficient between a *base* vector and another *vector*. The *base* vector is specified using the **–F** option. If the *base* or *vector* is not given, it is assumed to come from the standard input. Each *vector* is compared to the *base*. Both *base* and *vector* must be of the same length.

OPTIONS

—F*vector*
vector is the *base*.

EXAMPLES

The following example outputs the ordinary correlation coefficients between *vectors* A and B, and *vectors* A and C.

```
cor -FA B C
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

cp — copy

SYNOPSIS

cp [**—i**] [**—r**] *filename1 filename2*

cp [**—i**] [**—r**] *filename ... directory*

DESCRIPTION

Filename1 is copied onto *filename2*. The mode and owner of *filename2* are preserved if it already existed; the mode of the source file is used otherwise.

In the second form, one or more *filenames* are copied into the *directory* with their original file–names.

Cp refuses to copy a file onto itself.

OPTIONS

—i **Cp** will prompt the user with the name of the file whenever the copy will cause an old file to be overwritten. An answer of **y** will cause **cp** to continue. Any other answer will prevent it from overwriting the file.

—r If any of the source files are directories, **cp** copies each subtree rooted at that name; in this case the destination must be a directory.

EXAMPLES

The following invocation will copy all the files in the current directory beginning with the letters *xyz* to the directory *mhl*.

```
cp xyz* mhl
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Copying a directory without the **—r** option causes a file to be created which contains the same data as the directory, but which is not a directory. This can be useful for viewing the contents of a directory with **cat.**

The **—i** option for **cp** is not the same as that for **rm.**

SEE ALSO

cat(1), cpio(1), mv(1), pr(1), rcp(1n), rm(1), sh(1sh).

NAME

cpio — copy file archives in and out

SYNOPSIS

cpio **—i** [[**—B**] [**—F** *number*] [**—N** *size*] [**—P** *prompt*] [**—R** *path*] [**—S**] [**—V** *volume*] [**—b**] [**—c**] [**—d**] [**—f**] [**—m**] [**—n** *blocks*] [**—r**] [**—s**] [**—t**] [**—u**] [**—v**] [**—6**]] [*patterns*]

cpio **—o** [**—B**] [**—N** *size*] [**—P** *prompt*] [**—V** *volume*] [**—a**] [**—c**] [**—n** *blocks*] [**—v**]]

cpio **—p** [**—a**] [**—d**] [**—l**] [**—m**] [**—r**] [**—u**] [**—v**]] *directory*

DESCRIPTION

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(1sh)*. 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 * (for example, select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below.

This procedure is usually used to extract files from a **cpio** archive file or tape.

Cpio **—o** (copy out) reads the standard input to obtain a list of pathnames and copies those files onto the standard output together with pathname and status information. The normal procedure is to redirect the standard output to a file or device (such as a tape drive), thus archiving the files.

Cpio **—p** (pass) reads the standard input to obtain a list of pathnames of files that are conditionally created and copied into the destination *directory* tree based upon the options described below. This is similar to using *cp(1)* with the **—r** option.

Multi-volume Archives

Cpio with the options **—i** and **—o** has the ability to handle multi-volume files. The name of the device that data is to be read from or written to can be specified by the **—V** option. The number of 512-byte blocks on a volume can be specified with the **—n** option (if this is not specified, **cpio** goes until the system says the volume is full or empty).

When the volume is full or empty, the program enters an interactive mode and prints the message

No more (**data** or **space**) **on** (*Volume*).

followed by a short explanation, which may either be the default explanation or the text specified by the **—P** option.

If a device or filename was specified with the **—V** option, or if a device or filename was specified in interactive mode before, the same file or device can be used by typing return. If no device or file has been named, the name of the device or file to be used must be entered. (The exception to

this is when **cpio** was invoked to read or write a regular file. In this case, typing return will cause **cpio** to continue using the same file.) In case of disk or tape, this should be the same device name that was used when the command was invoked.

When using tapes or diskettes in this mode, you must have the new tape or diskette ready to read or write before you hit return.

This information is also supplied during interactive mode.

It is important to note that any type of file can be read or written in this mode. This makes it possible to break up archives into smaller pieces for easier transport.

Portability
Cpio archive headers contain the device numbers of the files on the archives. The standard device number is 16 bits wide, but it may be longer on some systems. This version of **cpio** handles this difference in a portable way. On systems with longer device numbers, device numbers for regular files are hashed so that they will fit in the header and still be unique for purposes of relinking. Device numbers for special files are marked specially and stored in the data area.

Because of this possible difference, **cpio** archives should not be extracted (with **—i**) by the superuser.

OPTIONS

—6 Process an old file (for example, UNIX System *Sixth* Edition format). Only useful with **—i** (copy in).

—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 tape or floppy disk.

—F *number*
Stop extracting or listing after *number* files have been extracted/listed. This is only usable with the **—i** option.

—N *size*
Input/output is to be blocked *size* times 512 bytes to the record (does not apply to the **pass** option; meaningful only with data directed to or from tape of floppy disk. The **—B** option is equivalent to **—N 10**. (See CAVEATS.)

—P *prompt*
Print the given text instead of the default media change text (see Multi-volume Archives above). The text is printed as given, with no special processing done.

—R *path*
Copy relative to *path*. The given *path* is prepended to all names beginning with a */*. This action takes place before the rename (**—r** option) process. This is only usable with the **—i** option.

—S
Swap half words. Use only with the **—i** option.

—V *volume*
Specifies the name of the file or device to read/write. Can not be used with **—p.**

—a Reset access and modification times of input files after they have been copied.

—b Swap both bytes and half words. Use only with the **—i** option.

—c Write *header* information in ASCII character form for portability.

—d *Directories* are to be created as needed.

—f Copy in all files except those in *patterns.*

—i Extracts files from the standard input which is assumed to be the product of a previous **cpio —o**.

—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.

—n *blocks*
Specifies the number of 512–byte blocks that can be read from or written to the volume (does not require the **—V** option). When this number is reached, the program enters interactive mode so that a new volume can be mounted. Cannot be used with the **—p** option.

—o Reads the standard input to obtain a list of pathnames and copies those files onto the standard output together with pathname and status information.

—p Reads the standard input to obtain a list of pathnames of files that are conditionally created and copied into the destination *directory* tree based upon the options.

—r Interactively renames files. If the user types a null line, the file is skipped.

—s Swap bytes. Use only with the **—i** option.

—t Print a *table of contents* of the input. No files are created.

—u Copy unconditionally (normally, an older file will not replace a newer file with the same name).

—v Verbose: causes a list of filenames to be printed. When used with the **—t** option, the table of contents looks similar to the output of an **ls —l** command (see *ls(1)*).

EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls | cpio -o >/dev/mt0

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

This next example shows a typical use of **cpio** to archive a directory and all subdirectories and place the archive in the file "arch.cpio".

```
find dir -print ¦ cpio -ov -V arch.cpio
```

The trivial case

```
find . -print | cpio -oB >/dev/rmt0
```

can be handled more efficiently by:

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

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	The user quit while changing volume.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

The output from the **—v** option is placed on the standard error. See the manual page for the shell being used for information on redirecting the standard error.

On some versions of UNIX, pathnames in **cpio** archives are restricted to 128 characters. This version restricts pathnames to the system maximum (currently 1024); thus, the resulting archives may be unportable.

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.

Cpio does not know about symbolic links, but since it is usually used with **find,** there is little danger of getting into loops. Also, instead of archiving or copying symbolic links, **cpio** copies the files pointed to by the links, if they exist.

Only the superuser can copy special files.

The editors **ex, vi, e, edit,** and **view,** and the programs **more** and **page** will not allow the editing or displaying of non–ASCII **cpio** archives. ASCII **cpio** archives are not restricted.

When using **cpio** with a streaming tape drive, the option **—N 256** should be used in order to keep the tape streaming.

SEE ALSO

ar(1), cp(1), file(1), find(1), tar(1), cpio(5).

NAME

csh — a shell (command interpreter) with C–like syntax

SYNOPSIS

csh [**—cefinstvVxX**] [*arg* ...]

DESCRIPTION

Csh is a first implementation of a command language interpreter incorporating a history mechanism (see *history(1csh)*) job control facilities (see *jobs(1csh)*), interactive command and filename completion, and a C–like syntax. So as to be able to use its job control facilities, users of **csh** must (and automatically) use the new tty driver fully described in *tty(4)*. This new tty driver allows generation of interrupt characters from the keyboard to tell jobs to stop. See *stty(1)* for details on setting options in the new tty driver.

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 crt's 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 &, |, ;, ⟨, ⟩, (, and) form separate words. If doubled in &&, ||, ⟨⟨, or ⟩⟩ these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with a backslash (\). A newline preceded by a '\' is equivalent to a blank.

In addition, strings enclosed in matched pairs of quotations marks, ', `, or ", form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of ' or " characters a newline preceded by a \ gives a true newline character.

When the shell's input is not a terminal, the character # introduces a comment which continues to the end of the input line. It is prevented this special meaning when preceded by \ and in quotations using `, ', and ".

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by ¦ (pipe) 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 a semicolon (;), and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for the sequence to terminate by following it with an &.

Any of the above may be placed in parentheses ( ) to form a simple command (which may be a component of a pipeline, and so forth). 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.)*

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; however, it only does so just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable **notify,** the shell will notify you immediately of changes of status in background jobs. There is also a shell command **notify** which marks a single process so that its status changes will be immediately reported. By default, **notify** marks the current process; simply say **notify** after starting a background job to mark it.

When you try to leave the shell while jobs are stopped, you will be warned with *You have stopped jobs.* You may use the **jobs** command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the suspended jobs will be terminated.

Substitutions

Described below are the various transformations the shell performs on the input in the order in which they occur.

History substitutions

See the manual page for *history(1csh)* for information on history substitutions.

Quotations with ' and "

The quotation of strings by single quotation marks (') and double quotation marks (") can be used to prevent all or some of the remaining substitutions. Strings enclosed in ' are prevented any further interpretation. Strings enclosed in " may be expanded as described below.

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

Alias substitution

See the manual page for *alias(1csh)* for information on alias substitutions.

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 quotation marks ("), where it *always* occurs, and within single quotation marks ('), where it *never* occurs. Strings quoted by open single quotes (') 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 " or given the **:q** modifier, the results of variable substitution may eventually be command and filename substituted. Within double quotation marks ("), a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variables value separated by blanks. When the **:q** modifier is applied to a substitution, the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or

filename substitution.

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

$*name*
${*name*}
Are replaced by the words of the value of variable *name,* each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character (_) is considered a letter.
If *name* is not a shell variable, but is set in the environment, then that value is returned (but a colon (:) modifiers and the other forms given below are not available in this case).

$*name[selector]*
${*name[selector]*}
Can be used to select only some of the words from the value of *name.* The selector is subjected to $ substitution and can consist of a single number or two numbers separated by a dash (—). 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** can be applied to the substitutions above, as can **: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 colon (:) modifier on each $ expansion.*

The following substitutions may *not* be modified with colon (:) 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.

$<
Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Command and filename substitution

The remaining substitutions-- command and filename substitution-- are applied selectively to the arguments of 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 open single quotation marks (‘). The output from such a command is normally broken into separate words at blanks, tabs, and newlines, with null words being discarded, and this text then replacing the original string. Within double quotation marks ("), 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 tilde 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 filenames which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing filename, but it is not required for each pattern to match. Only the metacharacters *, ?, and [imply pattern matching, while the characters ~ and { are more akin to abbreviations.

In matching filenames, the dot 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 a dash (—) matches any character lexically between, and including, the two characters. Note that if the characters

are reversed in the collating sequence (such as the pattern "[z–a]"), the dash is ignored.

The tilde character (~) at the beginning of a filename is used to refer to home directories. Standing alone, 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 dash (—) 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 tilde character (~) is followed by a character other than a letter or /, or appears anywhere but at the beginning of a word, it is left undisturbed.

The metanotation *a{b,c,d}e* is a shorthand for *abe ace ade.* Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus *~source/s1/{oldls,ls}.c* expands to */usr/source/s1/oldls.c /usr/source/s1/ls.c* whether or not these files exist without any chance of error if the home directory for *source* if */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 can be redirected with the following syntax:

< filename

Open *filename* (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.

> filename
>! filename
>& filename
>&! filename

The *filename* is used as standard output. If the file does not exist, it is created; if the file exists, it is truncated, with its previous contents being lost.

If the variable **noclobber** is set, then the file must not exist or be a

character special file (for example, 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. *Filename* is expanded in the same way as < input filenames are.

>> *filename*
>>& *filename*
>>! *filename*
>>&! *filename*

Uses *filename* 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 in–line data. This permits shell command scripts to function as components of pipelines and allows the shell to block–read its input. Note that the default standard input for a command run detached is *not* modified to be the empty file */dev/null*; rather the standard input remains as the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, then the process will block and the user will be notified (see *Jobs* above).

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

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, for example, *'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 a zero (0) are considered octal numbers. Null or missing arguments are considered zero. 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 braces { and } and file enquiries of the form *—l filename* 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.

Command/filename completion and file listing

In interactive shells, commands and filenames may be completed for you by using the character escape (^ []), and listed by using the character ^D (control–d).

Command completion is turned on by setting the shell variable "complete". When this variable is set, typing the escape character completes the command or filename up to the point where more than one name could match, or only one name matches. For example, if the current directory contains the files "hellothere" and "hellofolks", typing a command name followed by the letter 'h' followed by an escape will change the command line to "command hello" and the terminal bell will ring. By typing the letter 't' followed by an escape, the command line becomes "command hellothere". This works with command names as well (note: with the exception of the commands in the current directory, execute permission is not checked for).

The bell printed during ambiguities or unmatchable strings can be changed by setting the shell variable "vbell". If this is set to nothing, the visible bell sequence from the terminal capability entry (see *termcap(5t))* is used. If the variable is set to anything else, that string will be printed.

File listing is turned on by setting the shell variable "list". When this variable is set, typing ^D (control–d) lists all commands or files that match the current string being typed. Assume the same two files listed above. If the command line "command hello" is typed followed by a ^D, the shell will list the two filenames that begin with 'hello' in the current directory.

If the "list" variable is set to a string that begins with the letter 'f', the files are marked with a character that denotes the type of file, as with the command **ls –F** . If the "list" variable is set to a string that begins with the letter 'l', all symbolic links are listed with a trailing '@'. This works with command names, with the additional feature that if the shell variable "listpathnum" is set, the number of the directory in the "path" variable is listed along with the commands.

The special case in which the string begins with the character '~' causes the names of all users that match the string to be printed. For example, typing "command ~a" followed by a ^D will print all user names that begin with the letter 'a'.

There are some deficiencies with this feature. First, the names must match exactly. No metacharacters are expanded. Also, aliases are not expanded, though the user can set up a special directory in the execution path which contains (non–executable) files with the same names as the aliases.

It is important to note that setting either the "complete" or "list" variables causes the shell to require character echoing and cooked input mode, so changing these modes (such as with *stty(1)*) will be ineffective.

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.

Following is the syntax for the built–in control flow commands:

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, and 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; or if *expr2* is true then the commands to the second else are executed, and so forth. 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**.)

while *(expr)*
...
end

While the specified expression evaluates nonzero, the commands between the **while** and the matching end are evaluated. **Break** and **continue** can be used to terminate or continue the loop prematurely. (The **while** and **end** must appear alone on their input lines.) 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.

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. Prompting occurs here the first time through the loop as for the **while** command.

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 [...] can 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**.

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

Built–in commands

Built–in commands are executed within the shell. If a built–in command occurs as any component of a pipeline except the last, then it is executed in a subshell. The built–in commands are documented in separate manual entries. See the SEE ALSO section at the end of this document for a list of manual pages to see for descriptions of these commands. statement if the input is a terminal.

Non–built–in command execution

When a command to be executed is found to not be a builtin command the shell attempts to execute the command via *execve(2)*. Each word in the variable *pathname* 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 *pathname* which does not begin with a /, the shell concatenates with the given command name to form a pathname 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 Bourne shell commands and a copy of /bin/sh is spawned to read it.

It is important to note that this version of **csh** does not automatically execute scripts. In order to execute a file as a **csh** script, the first line of the command must be of the form "#!/bin/csh [options]". See *execve(2)* for more information.

Signal handling

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

OPTIONS

If argument 0 to the shell is a dash (—) 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 nonzero exit status.

—f The shell will start faster, because it will neither search for nor execute commands from the file *.cshrc* in the invoker's 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 aids 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 backslash (\) can 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 to the diagnostic output (even in cases of redirection) 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 were given the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by **$0**. Since many

systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute such a standard shell if the first character of a script is not a #; e.g., if the script does not start with a comment. Remaining arguments initialize the variable **argv**.

FILES

$HOME/.cshrc	Read at beginning of execution by each shell.
$HOME/.login	Read by login shell, after *.cshrc* at login.
$HOME/.logout	Read by login shell, at logout.
/bin/sh	Standard shell, default for running scripts. Scripts to be run by **csh** must begin with '#!/bin/csh'.
*/tmp/sh**	Temporary file for <<.
/etc/passwd	Source of home directories for ~ filenames.

VARIABLES

CDPATH	The change directory search path. Used to set *$cdpathname*.
HOME	The user's home directory. Used to set *$home*.
PATH	The execution search path. Used to set *$pathname*.
TERM	The type of terminal being used. Used in command completion.
TERMCAP	The name of the terminal capability file, or the capability entry itself.
SHELL	The user's login shell. Used to set *$shell*.
USER	The user's login name. Used to set *$user*.

CAVEATS

When a command is restarted from a **stop**, the shell prints the directory it started in, if this is different from the current directory; this can be misleading (for example, wrong) as the job may have changed directories internally.

Shell builtin functions are not stoppable/restartable. Command sequences of the form `a ; b ; c` are also not handled gracefully when stopping is attempted. If you suspend *b*, the shell will then immediately execute *c*. This is especially noticeable if this expansion results from an **alias**. It suffices to place the sequence of commands in parentheses ()'s to force it to a subshell; for example, `( 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 a pipe (¦), and to be used with & and ; metasyntax.

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

Symbolic links fool the shell. In particular, **dirs** and **cd ..** don't work properly once you've crossed through a symbolic link.

Words can be no longer than 2048 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves filename expansion is limited to 1/6'th 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.

Input/output redirection occurs even if *expr* is false, and when *commmand* is not executed.

When command completion or file listing are turned on, terminal echo mode will always be turned on and the terminal will always be set to cooked input mode. This is done to ensure that command completion and file listing behave properly. If you need to change these terminal characteristics, you must turn off command completion and file listing.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), access(2), execve(2), fork(2), killpg(2), pipe(2), sigvec(2), umask(2), setrlimit(2), wait(2), tty(4), a.out(5), environ(7).

REFERENCES

Introduction to the C-shell in the *UTek Tools* documentation.

NAME

csplit — context split

SYNOPSIS

csplit [—**c**] [—**s**] [—**k**] [—**f** *prefix*] *filename arg1* . . . [*argn*]

DESCRIPTION

Csplit reads *filename* and separates it into *n* + 1 sections, defined by the arguments *arg1*. . . *argn*. By default the sections are placed in *xx*00 . . . *xx n* (*n* may not be greater than 99). These sections get the following pieces of *filename*:

00: From the start of *filename* 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 *filename*.

OPTIONS

—**c** Print the number of files created instead of the sizes of the files. If the —**s** option is also given, the —**c** option is turned off.

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

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

—**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.

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 (for example, **/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* (line number). 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 newlines. **Csplit** does not affect the original file; it is the user's responsibility to remove it.

EXAMPLES

This example creates four files, *cobol100 . . . cobol103*:

```
csplit -f cobol file  '/procedure division/'

/par5./  /par16./
```

After editing the *split* files, they can be recombined as follows:

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

Note that this example overwrites the original file.

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

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

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

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

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

ed(1), sh(1sh).

NAME

ctags — create a *tags* file

SYNOPSIS

ctags [**—B**] [**—F**] [**—a**] [**—t**] [**—u**] [**—v**] [**—w**] [**—x**] *filename ...*

DESCRIPTION

Ctags makes a tags file for *ex(1)* from the specified C, Pascal, and FORTRAN sources. A *tags* file gives the locations of specified objects (in this case functions and typedefs) in a group of files. Each line of the *tags* file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern, typedefs with a line number. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the *tags* file, the editting system **ex** can quickly find these object definitions.

Files whose name ends in *.c* or *.h* or have no suffix are assumed to be C source files and are searched for C routine and macro definitions. Others are first examined to see if they contain any Pascal or FORTRAN routine definitions; if not, they are processed again looking for C 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.

OPTIONS

—a Append to *tags* file.

—t Create *tags* for typedefs.

—u 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. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.)

—v An index of the form expected by *vgrind* (see CAVEATS) is produced on the standard output. This listing contains the function name, filename, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through **sort —f** (see EXAMPLES).

—w
Suppresses warning diagnostics.

—x **Ctags** produces a list of object names, the line number and filename on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off–line readable function index.

—B
Use backward searching patterns (?...?).

—F Use forward searching patterns (/.../) (default).

EXAMPLES

This example produces a *tags* file for a program made up of the source files *example.c* and *extras.c*. The entries in the *tags* file will have backward search for patterns:

```
ctags -B example.c extras.c
```

Executing the command **vi —t main** will edit the file containing the routine main( ), and place the cursor at the definition of that routine by searching backwards.

The following command set prints a sorted cross reference listing for the files using *vgrind* (see CAVEATS):

```
ctags -v files ¦ sort -f › index
vgrind -x index
```

FILES

tags	*Tags* file for use with *ex(1)*.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Recognition of functions, subroutines, and procedures for FORTRAN and Pascal is done in a very simple way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name, only the first is tagged.

The method of deciding whether to look for C or Pascal and FORTRAN functions uses the prefix of the file, and may not always decide on the correct language.

Does not process #ifdefs (used by the preprocessor).

Should know about Pascal types. Relies on the input being well formed to detect typedefs. Use of **—tx** shows only the last line of typedefs.

The —**v** option is supported for completeness. The utility **vgrind** may not be available on your system.

SEE ALSO

cflow(1), cxref(1), ex(1), sort(1), vi(1).

NAME

tip, cu — connect to a remote system

SYNOPSIS

tip [**—v**] [*—speed*] *system-name*
tip [**—v**] [*—speed*] *phone-number*
cu *phone-number* [**—t**] [**—s** *speed*] [**-a** *acu*] [**—l** *line*] [**—#**]

DESCRIPTION

Tip and **cu** establish a full–duplex connection to another machine, giving the appearance of being logged in directly on the remote CPU. It goes without saying that you must have a login on the machine (or equivalent) to which you wish to connect. The preferred interface is **tip**. The **cu** interface is included for those people attached to the **call unix** command of version 7 UNIX. This manual page describes only **tip**.

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well). A tilde (~) appearing as the first character of a line is an escape signal; the following are recognized:

~⟨CTRL–D⟩~.
: Drop the connection and exit (you may still be logged in on the remote machine).

~**c** [*name*]
: Change directory to *name* (no argument implies change to your home directory).

~**!**
: Escape to a shell; (exiting the shell will return you to **tip**).

~⟩
: Copy file from local to remote. **Tip** prompts for the name of a local file to transmit.

~⟨
: Copy file from remote to local. **Tip** prompts first for the name of the file to be sent, then for a command to be executed on the remote machine.

~**p** *from* [*to*]
: Send a file to a remote UTek or UNIX host. The **put** command causes the remote UTek or UNIX system to run the command string **cat** ⟩ *to*, while **tip** sends it the *from* file. If the *to* file isn't specified, the *from* filename is used. This command is actually a UTek (UNIX) specific version of the ~⟩ command.

~**t** *from* [*to*]
Take a file from a remote UTek or UNIX host. As in the **put** command, the *to* file defaults to the *from* filename if it isn't specified. The remote host executes the command string

```
cat from;echo ⟨CTRL-A⟩
```

to send the file to **tip**.

~| Pipe the output from a remote command to a local UTek process. The command string sent to the local UTek system is processed by the shell.

~**#** Send a **BREAK** to the remote system.

~**s** Set a variable (see the discussion below).

~⟨CTRL-Z⟩
Stop **tip** (only available with job control).

~{ Receive a text file from the remote host using the XMODEM protocol. Must issue the appropriate command to start XMODEM transfer *before* giving this escape to *tip*. Translation is performed from CP/M file format (CR/LF) to UTek text file format (LF). If **beautify** is set then all bytes have the parity bit removed for consistency with UTek editors.

~} Send a text file to the remote host using the XMODEM protocol. Translation is done from UTek text file format to CP/M format as dictated by the protocol. Must issue the XMODEM command on the remote host first.

~**(** Receive a binary file from the remote host using the XMODEM protocol. No translation is performed.

~**)** Send a binary file to the remote host using the XMODEM protocol. No translations are done, the file is sent as is. The protocol dictates that the last 128 byte sector be padded with control Z characters, so this may not be suitable for transfer between UTek/UNIX hosts.

~**?** Get a summary of the tilde escapes.

Tip uses the file */etc/remote* to find how to reach a particular system and to find out how it should operate while talking to the system; refer to *remote(5n)* for a full description. Each system has a default baud rate with which to establish a connection. If this value is not suitable, the baud rate to be used may be specified on the command line; for example, **tip -300 mds**.

When **tip** establishes a connection it sends out a connection message to the remote system; the default value, if any, is defined in */etc/remote.*

When **tip** prompts for an argument (for example, during setup of a file transfer) the line typed may be edited with the standard **erase** and **kill** characters. A null line in response to a prompt, or an interrupt, will abort the dialogue and return you to the remote machine.

Tip guards against multiple users connecting to a remote system by opening modems and terminal lines with exclusive access, and by honoring the locking protocol used by *uucp(1n).*

During file transfers **tip** provides a running count of the number of lines transferred. When using the ~〉 and ~〈 commands, the **eofread** and **eofwrite** variables are used to recognize end–of–file when reading, and specify end–of–file when writing (see below). File transfers normally depend on tandem mode for flow control. If the remote system does not support tandem mode, **echocheck** may be set to indicate **tip** should synchronize with the remote system on the **echo** of each transmitted character.

When **tip** must dial a phone number to connect to a system, it will print various messages indicating its actions. **Tip** supports the Racal–Vadic 831 auto–call–units; the Ventel 212+, Racal–Vadic 3451, and Bizcomp 1031 and 1032 integral call unit/modems.

INSTALLATION

Files needed are:

Name	Permissions	Owner
/bin/tip	–rwsr–xr–x	uucp
/etc/remote	–rw–rw–r––	sys
/usr/spool/uucp	drwxr–xr–x	uucp

Optional files are:

/etc/phones	–rw–rw–r––	sys
~/.tiprc		
~/.tipphones		
~/.tipremote		

FOR DIRECT RS-232-C CONNECTION

You need the following equipment:

An RS-232-C line to a target system. (Make sure you are able to login to the target system.)

A modem adapter cable (Tek part # 012-1120-00). It has two male ends that you plug into the female connectors on the target system and the workstation. It also switches the control lines so that two DCE ports can communicate as if they were a DCE/DTE pair.

Things to do:

1. Disable logins on the port you are going to use on your workstation by changing the entry in the */etc/ttys(5)* file. Then restart the init process by rebooting or sending it a hangup signal. For example make the following changes in the */etc/ttys* file on your workstation:

change: 1ytty1 to 0ytty1

2. Type: **kill -1 1**. This sends the init process the hangup signal. *Init(8)* rereads the */etc/ttys* file and turns off the login on port tty1. This prevents a **login** process from interfering with the port you are about to use for **tip**.

3. Use **chown** to give the tty port you are going to use for the **tip** connection to uucp. For example:

/etc/chown uucp /dev/tty1

4. Put an entry in the */etc/remote* file that describes the port you are going to use. For example:

```
direct¦direct 9600 baud line:\
    :dv=/dev/tty1:br#9600:ta:ie=^A\
    :oe=^A
```

5. Using the modem adapter cable, connect the login line from the target machine to the port you have chosen on your workstation.

6. Type **tip direct**, and **tip** will open a 9600 baud connection to the target host if you use the above examples.

FOR MODEM CONNECTION:

You need the following equipment:

A modem adapter cable as above.

A modem that is supported by the uucp Automatic Calling Unit library. Modems currently supported include: BIZCOMP, VENTEL 212+, VADIC 831 RS-232-C adaptor, VADIC 3451, HAYES smart modem. Other modems will work if they have an emulation mode for one of the above modems.

To make the connection perform the following steps:

1. Disable logins on the port you are going to use on your workstation by changing the entry in the */etc/ttys* file. Then restart the **init** process by rebooting or sending it a hangup signal. For example make the following changes in the */etc/ttys* file on your workstation. This example is for tty1:

change **1ytty1** to **0ytty1**

2. Use **chown** to give the tty port you are going to use for the **tip** connection to uucp. For example:

/etc/chown uucp /dev/tty1

3. Put an entry in the */etc/remote(5n)* file that describes the port and modem you are going to use. For example:

```
dial1200|1200 Baud Hayes :\
    :dv=/dev/tty1:br#1200:du:at=hayes:
```

4. Connect your modem to your workstation using the modem adapter cable.

5. Invoke **tip**. For example:

tip dial1200 5551212

Sometimes you need to type a carriage return after the "connected" message to get a prompt.

OPTIONS

—v This option causes **tip** to display the setting of its variables as they are done.

FILES

/etc/remote	Global system descriptions.
/etc/phones	Global phone number database.
${REMOTE}	Private system descriptions.
${PHONES}	Private phone numbers.
~/.tiprc	Initialization file.
*/usr/spool/uucp/LCK..**	Lock file to avoid conflicts with **uucp**.

DIAGNOSTICS

These are the most common messages. There are many others.

link down **Tip** displays this message when it cannot open the RS–232–C port. This will happen if a cable is not plugged in or if the cable that is used does not have the *carrier detect* pin connected.

all ports busy This message is displayed when a lock file is present in the */usr/spool/uucp* directory for the port **tip** is trying to use. This means that some other user is using this port and **tip** is locked out for the time being. If a lock file is present inadvertently, and there really is no one else trying to use this port, remove the lock file. You will need to run as superuser to do this.
Example:

```
rm /usr/spool/uucp/LCK..tty1
```

VARIABLES

Tip maintains a set of *variables* which control its operation. Some of these variable are read–only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the **s** escape. The syntax for variables is patterned after *vi(1)* and *mail(1mh)*. Supplying **all** as an argument to the set command displays all variables readable by the user. Alternatively, the user may request display of a particular variable by attaching a ? to the end. For example, **escape?** displays the current escape character.

Variables are numeric, string, character, or Boolean values. Boolean variables are set merely by specifying their name; they may be reset by prepending a ! to the name. Other variable types are set by concatenating an equal sign(=) and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables. Variables may be initialized at run–time by placing set commands (without the ~**s** prefix in a file *.tiprc* in your home directory). The **—v** option causes **tip** to display the sets as they are made. Certain common variables have

abbreviations. The following is a list of common variables, their abbreviations, and their default values:

beautify
(Bool) Discard unprintable characters when a session is being scripted; abbreviated **be**.

baudrate
(num) The baud rate at which the connection was established; abbreviated **ba**.

dialtimeout
(num) When dialing a phone number, the time (in seconds) to wait for a connection to be established; abbreviated **dial**.

echocheck
(Bool) Synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted; default is **off**.

eofread
(str) The set of characters which signify and end-of-transmission during a ~< file transfer command; abbreviated **eofr**.

eofwrite
(str) The string sent to indicate end-of-transmission during a ~> file transfer command; abbreviated **eofw**.

eol
(str) The set of characters which indicate an end-of-line. **Tip** will recognize escape characters only after an end-of-line.

escape
(char) The command prefix (escape) character; abbreviated **es**; default value is a tilde (~).

exceptions
(str) The set of characters which should not be discarded due to the beautification switch; abbreviated **ex**; default value is \t\n\f\b.

force
(char) The character used to force literal data transmission; abbreviated **fo**; default value is ⟨CTRL-P⟩.

framesize
(num) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated **fr**.

host
(str) The name of the host to which you are connected; abbreviated **ho**.

prompt

(char) The character which indicates an end–of–line on the remote host; abbreviated **pr**; default value is **\n**. This value is used to synchronize during data transfers. The count of lines transferred during a file transfer command is based on receipt of this character.

raise

(Bool) Uppercase mapping mode; abbreviated **ra**; default value is **off**. When this mode is enabled, all lowercase letters will be mapped to uppercase by **tip** for transmission to the remote machine.

raisechar

(char) The input character used to toggle uppercase mapping mode; abbreviated **rc**; default value is ⟨CTRL–A⟩.

record

(str) The name of the file in which a session script is recorded; abbreviated **rec**; default value is **tip.record**.

script

(Bool) Session scripting mode; abbreviated **sc**; default is **off**. When **script** is **true**, **tip** will record everything transmitted by the remote machine in the script record file specified in **record**. If the **beautify** switch is on, only printable ASCII characters will be included in the script file (those characters between 040 and 0177) [also on XMODEM text file receives].

tabexpand

(Bool) Expand tabs to spaces during file transfers; abbreviated **tab**; default value is **false**. Each tab is expanded to eight spaces.

verbose

(Bool) Verbose mode; abbreviated **verb**; default is **true**. When verbose mode is enabled, **tip** prints messages while dialing, shows the current number of lines transferred during a file transfer operations, and more.

SHELL

(str) The name of the shell to use for the ~! command; default value is */bin/sh*, or taken from the environment.

HOME

(str) The home directory to use for the ~**c** command; default value is taken from the environment.

EXAMPLES

This is how variables are set up in the */etc/remote* file:

```
direct|direct 9600 baud line:\
    :dv=/dev/tty1:br#9600:ta:ie=^A\
    :oe=^A
```

The *ie* and *oe* strings refer to the **eofread** and **eofwrite** strings described above. See the *remote(5n)* man page for more details.

This is how variables are set up using a tilde *s* escape. When **tip** answers "W~[set]" it is printed over your ~**s**. The escape character must be the first character typed on a line.

You type:

```
$~s
```

tip types:

```
~[set]
```

You type:

eofr=end_of_file_read_string

RETURN VALUE

[0]	No errors.
[nonzero]	Errors occurred.

CAVEATS

The full set of variables is undocumented and should probably be pared down.

SEE ALSO

remote(5n), phones(5n).

NAME

cusum — cumulative sum

SYNOPSIS

cusum [**—c***n*] [*vector ...*]

DESCRIPTION

Output is a vector with the *i*th element being the sum of the first *i* elements from the input *vector*. If more than one *vector* is given, **cusum** gives the sum of each vector in turn (*not* the sum of all vectors). If no *vector* is given, the standard input is assumed.

OPTIONS

—c*n*

n is the number of output elements per line.

EXAMPLES

The following example outputs the cumulative sum of the elements of A, three per line.

```
cusum -c3 A
```

The following example outputs the cumulative sum of the elements of A followed by the cumulative sum of the elements of B.

```
cusum A B
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

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

SYNOPSIS

cut —**c***list* [*filename ...*]
cut —**f***list* [—**d***char*] [—**s**] [*filename ...*]

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 (for example, 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 output is printed on the standard output.

Use *grep (1)* to make horizontal "cuts" (by context) through a file, or *paste (1)* to put files together column–wise (for example, horizontally). To reorder columns in a table, use **cut** and **paste .**

OPTIONS

list A comma separated list of field or column numbers that is used with the —**c** and —**f** options. The *list* is specified like the —**o** option of **nroff/troff** for page ranges; for example: **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 (for example, —**c1—72** would pass the first 72 characters of each line).

—**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.

—**f***list*
The *list* following —**f** is a list of fields assumed to be separated in the file by a delimiter character (see —**d**). For example, —**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.

—**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.

EXAMPLES

```
cut -d: -f1,5 /etc/passwd
```

This next example maps user IDs to *names* to set *name* to current loginname:

```
name=who am i ¦ cut -f1 -d" "
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Line length is limited to 1023 characters. An error will be produced if a line of greater length is encountered.

SEE ALSO

awk(1), comm(1), egrep(1), fgrep(1), grep(1), join(1), look(1), paste(1), sort(1), uniq(1).

NAME

cvrtopt — options converter

SYNOPSIS

cvrtopt [**—f***string*] [**—i***string*] [**—s***string*] [**—t***string*] [*arg ...*]

DESCRIPTION

Cvrtopt reformats *args* (usually the command line arguments of a calling shell procedure) to facilitate processing by shell procedures. An *arg* is either a filename (a string not beginning with a '–', or a '–' by itself) or an option string (a string of options beginning with a '–'). Output is of the form:

–option –option . . . filename(s)

All options appear singularly and preceding any filenames. Option names that take values (e.g., –r1.1) or are two letters long must be described through options to **cvrtopt**. Output is to the standard output.

Cvrtopt is usually used with *set(1sh)* in the following manner as the first line of a shell script:

```
set - `cvrtopt [=option(s)] $@`
```

Set resets the command argument string ($1,$2,...) to the output of **cvrtopt**. The minus option to **set** turns off all flags so that the options produced by **cvrtopt** are not interpreted as options to **set**.

OPTIONS

—f*string*
String accepts floating point numbers as values.

—i*string*
String accepts integers as values.

—s*string*
String accepts string values.

—t*string*
String is a two letter option name that takes no value.

Each *string* is a one or two letter option name.

EXAMPLES

The following example outputs **–c1 –x –y –ofile –r1.1e3 –xi A – B**

```
cvrtopt =ic,so,txi,fr –c1 –x,y,ofile A – B –r1.1e3,xi
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), getopt(1), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), set(1sh), sh(1sh), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), getopt(3), and *gps(5g).*

NAME

cxref — generate C program cross reference

SYNOPSIS

cxref [**—c**][**—D***name*][**—D***name* = *def*][**—I***dir*][**—o***filename*][**—s**][**—t**][**—U***name*][**—w**<*name*>] *filename* . . .

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.

OPTIONS

—c Print a combined cross–reference of all input files.

—o*filename*
Direct output to *filename*.

—s Operate silently; does not print input filenames.

—t Format listing for 80–column width.

—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.

—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.

—I*dir*
#include files whose names do not begin with a slash (*/*) are always sought first in the directory of the *filename* argument, then in directories named in **—I** options, then in directories on a standard list.

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

EXAMPLES

This example of **cxref** outputs the file *skip* in a width of 70 decimal columns:

```
cxref -w<70> skip
```

FILES

/usr/local/lib/xcpp	Special version of C–preprocessor
/usr/local/lib/xpass	Special version of lint

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files, anyway.

SEE ALSO

cc(1), lint(1).

NAME

date — print and set the date

SYNOPSIS

date [**—c**] [**—r**] [**—u**] [**—z** *zone*] [*[yy]mmddhhmm[.ss]*] [*+format*]

DESCRIPTION

The current date and time are printed or set (only the superuser may do the latter). 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.

The options **—c** and **—r** only have effect when setting the date. They are used to set the correction factor in the clock to make the clock more accurate. If neither option is used the correction factor is not changed.

The **—r** option resets the correction factor to 1.0. (The correction factor will have no effect) The superuser would typically set the date using the **—r** option when first setting up the machine.

The **—c** option is used to compute a correction factor if the clock is inaccurate. The superuser would usually use the **—c** option after the machine has run for a relatively long time (e.g. at least one day) and the clock has gained or lost a significant amount (e.g. at least several seconds) of time. The correction factor affects the time of day according to the following formula:

```
time_of_day = base_time + (elapsed_time * correction_factor)
```

If the environment variable TZNAME is set, its value is used for the time zone when it is printed. If TZNAME contains a comma, the text before the comma is used for standard time and the text after the comma is used for daylight time. For example, if TZNAME is set to "Pacific Standard Time,Pacific Daylight Time", the date "June 11, 1984 at 4:30 pm" would be printed as "Mon Jun 11 16:30:00 Pacific Daylight Time 1984".

The time zone may be set when setting the date by using the **—z** option. The *zone* may either be the number of hours west of GMT or the time zone name. The time zone name may be upper case, lower case, or mixed. This table shows the acceptable values for *zone*.

Zone	**Acceptable Values**		
Eastern European	–2	EET	EET DST
Middle European	–1	MET	MET DST

Western European	0	WET	WET DST
Atlantic	4	AST	ADT
Eastern	5	EST	EDT
Central	6	CST	CDT
Mountain	7	MST	MDT
Pacific	8	PST	PDT
Aust: Eastern	−10	AEST	AEST
Aust: Central	−10.5	ACST	ACST
Aust: Western	−8	AWST	AWST

Only these values are recognized. This option is ignored if the date is not being set.

If there is an argument after the option that 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 terminated with a new–line character, unless the final character in the format is a lone **%**.

Field Descriptors:

- **n** insert a new–line character
- **t** insert a tab character
- **D** date as mm/dd/yy
- **m** month of year — 01 to 12
- **d** day of month — 01 to 31
- **y** last 2 digits of year — 00 to 99
- **h** abbreviated month — Jan to Dec
- **a** abbreviated weekday — Sun to Sat
- **w** day of week — Sunday = 0
- **j** Julian date — 001 to 366
- **T** time as HH:MM:SS
- **H** hour — 00 to 23
- **M** minute — 00 to 59
- **S** second — 00 to 59
- **r** time in AM/PM notation
- **Z** time zone

OPTIONS

—c will compute a new correction factor using the formula:

```
diff = time_of_day - new_time
if (diff>0) /* clock is fast */
   correction_factor = elapsed_time / (diff + elapsed_time)
if (diff<0) /* clock is slow */
   correction_factor = -diff + elapsed_time / elapsed_time
```

—r Reset the correction factor to 1.0.

—u Universal time. The time printed is Greenwich Mean Time (GMT).

—z *zone*
Use the given zone when setting the date.

EXAMPLES

The following command

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

would generate output like :

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

FILES

/usr/adm/wtmp	The new date is written to this file if the date is set.

VARIABLES

TZNAME	Time zone name. Either a single name or a comma–separated pair.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

cal(1), gettimeofday(2), ctime(3c).

NAME

dc — desk calculator

SYNOPSIS

dc [*filename*]

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.

bc This is a preprocessor for **dc** providing infix notation and a C—like syntax which implements functions and reasonable control structures for programs.

c All values on the stack are popped.

d The top value on the stack is duplicated.

f All values on the stack are printed.

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.

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.

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.

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.

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.

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.

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.

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.

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.

z The stack level is pushed onto the stack.

Z replaces the number on the top of the stack with its length.

! interprets the rest of the line as a command to be executed by *sh(1sh)*.

? A line of input is taken from the input source (usually the terminal) and executed.

; : are used by *bc* for array operations.

x ⟩ *x* = *x*
The top two elements of the stack are popped and compared. Register *x* is executed if they obey the stated relation.

[...]
puts the bracketed ASCII string onto the top of the stack.

EXAMPLES

An example which prints the first ten values of n! is

```
dc
[la1+dsa*pla10>y]sy
0sa1
lyx
q
```

DIAGNOSTICS

dc : x is unimplemented where x is an octal number.
dc : Stack empty for not enough elements on the stack to do what was asked.
dc : Out of memory when the free list is exhausted (too many digits).
Nesting Depth for too many levels of nested execution.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[INTERNAL]	An unexpected error occurred. Execution was terminated. Record the message and save the core file for analysis. Contact service personnel at your Tektronix field office.

CAVEATS

If the scale factor is greater than 63, the multiplication error can be as big as 199.

SEE ALSO

bc(1), sh(1sh).

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.

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** or * to indicate a product.

Cbs is used only if *ascii, unblock, ebcdic, ibm,* or *block* conversion is specified. In the first two cases, *cbs* characters are placed into the conversion buffer, any specified character mapping is done, trailing blanks trimmed and new–line added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer, and blanks added to make up an output record of size *cbs*.

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

OPTIONS

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 copy need be done

cbs = *n*
: conversion buffer size

conv = ascii
: convert EBCDIC to ASCII

ebcdic
: convert ASCII to EBCDIC

ibm
: slightly different map of ASCII to EBCDIC

block
: convert variable length records to fixed length

unblock
: convert fixed length records to variable length

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

count = *n*
copy only *n* input records

files = *n*
copy *n* input files before terminating (makes sense only where input is a magtape or similar device).

ibs = *n*
input block size *n* bytes (default 512)

if = input file name; standard input is default

obs = *n*
output block size (default 512)

of =
output file name; standard output is default

seek = *n*
seek *n* records from beginning of output file before copying

skip = *n*
skip *n* input records before starting copy

sync
pad every input record to *ibs*

EXAMPLES

For example, to read an EBCDIC tape blocked ten 80–byte EBCDIC card images per record into the ASCII file *x:*

```
dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magtape. **Dd** is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

DIAGNOSTICS

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

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

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.

SEE ALSO

cp(1), tr(1).

NAME

assign, deassign — assign or deassign devices in a class

SYNOPSIS

assign [**—f**] [**—r**] class ...
deassign class ...

DESCRIPTION

The *assign* and *deassign* commands use the file */etc/assign.classes* to determine what devices are assignable and what class(es) they belong to. Assignable devices are divided into classes (e.g. mag tape drive 0) so that all versions of a device can be assigned together (raw and cooked versions for example). These classes, in turn, may be grouped into generic classes (e.g. any mag tape drive). These groupings are set up by the system administrator, and thus are installation dependant.

Assign changes the owner of the device(s) in the requested *class* to the current UID. If the requested class is a generic one, the first specific class found that is available is used. The device(s) assume protection mode of read and write by owner only. If more than one class is listed, the first one that contains an assignable device is used and the remainder are ignored. This allows the user to specify the order that classes will be checked.

The name of the specific class that was assigned is typed on standard output. If no devices are available in the class(es) desired, an error message is typed on standard error, and the program exits with return code 1, unless the **—r** option is used.

Deassign undoes the work of *assign* by releasing the device to the assignable pool. The special form **deassign all** will deassign all devices you have.

All devices are considered available to the superuser. If the log file does not exist, logging is ignored. The owner of devices that are free for assignment is the owner of the file */etc/assign.classes*.

OPTIONS

—f Does not prompt whether to deassign when used in conjunction with the **—r** flag. This allows *assign* **—r** to be used in a shell script.

—r Reassigns the devices of a requested class after determining that no requested class is available without overriding an assignment. Mail is sent to the user with the previous assignment. Unless used with the **—f** flag, a prompt is issued to determine whether to override the previous assignment of the device(s).

EXAMPLES

assign mt1 mt0

would try to assign class *mt1*, and would try for *mt0* only if *mt1* is unavailable.

Typical usage might be (using the Bourne shell):

```
mt=`assign mt`
while test ! "${mt}"
do sleep 120; mt=`assign mt`; done
```

do whatever needs doing on the mag tape using device names:

/dev/${mt}0 for 800 BPI cooked
/dev/${mt}1 for 1600 BPI cooked
/dev/r${mt}0 for 800 BPI raw
/dev/r${mt}1 for 1600 BPI raw
/dev/n${mt}0 for 800 BPI cooked, no rewind on close
/dev/n${mt}1 for 1600 BPI cooked, no rewind on close
/dev/nr${mt}0 for 800 BPI raw, no rewind on close
/dev/nr${mt}1 for 1600 BPI raw, no rewind on close

This assumes that the system administrator has set up a generic class name in */etc/assign.classes* of *mt* and a specific class names of form *mt0*, *mt1*, *mt2* ... and that the tape versions of the tape drives are named /dev/mt00, /dev/mt01, /dev/rmt00 ... The shell variable *mt* will have the name of the assigned tape drive (e.g. *mt0*, *mt1*, *mt2* ...)

FILES

/etc/assign.classes	system file containing class specifications
/usr/adm/devicelog	log file for device usage

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	No available class was found.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

assign.classes(5).

NAME

delta — make a delta (change) to an SCCS file

SYNOPSIS

delta [—r*SID***]** [**—s**] [**—n**] [**—g***list*] [**—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*(1sccs) (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. 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.

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin*(1sccs)) that may be present in the SCCS file (see **—m** and **—y** keyletters below).

OPTIONS

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*(1sccs)). 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*(1sccs) 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*(1sccs)) 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).

*MR*s 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*(1sccs)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the *MR* numbers. If a non–zero exit status is returned from *MR* number validation program, **delta** terminates (it is assumed that the *MR* numbers were not all valid).

—y*comment*
Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

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

—p Causes **delta** to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

d-file	Created during the execution of **delta**; removed after completion of **delta**.
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**.

DIAGNOSTICS

Use *sccshelp*(1sccs) for explanations.

CAVEATS

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.

SEE ALSO

admin(1sccs), bdiff(1sccs), get(1sccs), sccshelp(1sccs), prs(1sccs), sccsfile(5sccs).

NAME

deroff — remove nroff and tbl constructs

SYNOPSIS

deroff [**—w**] [**—mm**] [**—ms**] [**—ml**] [*filename*] ...

DESCRIPTION

Deroff reads each file in sequence and removes all **nroff** command lines, backslash constructions, macro definitions, **neqn** constructs, and table descriptions. The remaining text is then written on standard output.

Deroff will follow chains of included files ('.so' and '.nx' commands).

If no input file is given, **deroff** reads from the standard input file.

OPTIONS

—ml
Will remove lists (implies **—mm**).

—mm
Causes **mm** macros to be interpreted so that just sentences are output.

—ms
Causes **ms** macros to be interpreted so that just sentences are output.

—w
The output is a word list, one 'word' (string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed) per line, and all other characters ignored. Otherwise, the output follows the original, with the deletions mentioned above.

EXAMPLES

The use of the following command will remove **ms** macros from the file report, and print the results on standard output:

```
deroff -ms report
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Deroff is not a complete **nroff** interpreter, so it can be confused by subtle constructs. Most errors result in too much rather than too little output.

The **—ml** option applies to list macros in the **mm** macro package. Therefore, if this option is given, the **—mm** option is also turned on.

Though multiple **—m** options may be given, the last one given is the one that sets the macro package being used.

SEE ALSO

neqn(1), nroff(1), tbl(1).

NAME

df — disk free

SYNOPSIS

df [**—i**] [*filesystem...*][*filename...*]

DESCRIPTION

Df prints out the amount of free disk space available on the specified *filesystem,* e.g. "/dev/rp0a", or on the filesystem in which the specified *file,* e.g. "$HOME", is contained. If no file system is specified, the free space on all of the normally mounted file systems is printed. The reported numbers are in kilobytes.

There are two output formats. The following columns are listed normally:

```
Filesystem  kbytes  used  avail  capacity  Mounted on
```

The *Filesystem* column lists the name of the device for the filesystem, such as */dev/hp0a.* The *kbytes* column lists the number of kilobytes on the device. The *used* column lists the number of kilobytes allocated to files. The *avail* column lists the number of kilobytes still available for files. The *capacity* column lists the percentage of the filesystem space that is allocated. The *Mounted on* column lists the name of the root of the filesystem, such as /.

When the **—i** option is given, the following columns are also given:

```
iused   ifree  %iused
```

The *iused* column lists the number of inodes allocated. The *ifree* column lists the number of inodes still available. The *%iused* column lists the percentage of the total number of inodes that are allocated.

OPTIONS

—i Causes df to also report the number of inodes used and free.

EXAMPLES

The following example prints a report of the free and used disk space and inodes in the file system /tmp.

```
df -i /tmp
```

FILES

/etc/fstab	File containing the list of normally mounted filesystems.
/etc/mtab	File containing the mounted filesystem table.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

The *capacity* of a file system can be reported as more than 100% because swap space may be being used. The swap space is not normally included in the statistics.

SEE ALSO

fstab(5), mtab(5), icheck(8), quot(8).

NAME

diff — differential file and directory comparator

SYNOPSIS

diff [—**l**] [—**r**] [—**s**] [—**S***filename*] [—{**c,e,f,h**}] [—**b**] *dir1 dir2*
diff [—{**c,e,f,h**}] [—**b**] *file1 file2*
diff [—**D***string*] [—**b**] *file1 file2*

DESCRIPTION

In the first form of the command line, **diff** sorts the contents of the directories by name, and then runs the regular file **diff** algorithm (described below) on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed.

In the second and third forms, and when comparing text files which differ during directory comparison, **diff** tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, **diff** finds a smallest sufficient set of file differences. If neither *file1* nor *file2* is a directory, then either may be given as '—', in which case the standard input is used. If *file1* is a directory, then a file in that directory whose file–name is the same as the file–name of *file2* is used (and vice versa).

There are several options for output format; the default output format contains lines of these forms:

n1 a *n3,n4*
n1,n2 d *n3*
n1,n2 c *n3,n4*

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging 'a' for 'd' and reading backward one may ascertain 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 '>'.

OPTIONS

Options when comparing directories are:

—**l** Long output format; each text file **diff** is piped through *pr*(1) to paginate it, other differences are remembered and summarized after all text file differences are reported.

—**r** Causes application of **diff** recursively to common subdirectories encountered.

—**s** Causes **diff** to report files which are the same, which are otherwise not mentioned.

—**S***filename*
Starts a directory **diff** in the middle beginning with *filename*.

Except for **—b,** which may be given with any of the others, the following options are mutually exclusive:

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

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

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

```
\v'-1p' (shift; cat $*; echo '1,$p') | ed — $1
```

Extra commands are added to the output when comparing directories with **—e,** so that the result is a *sh(1sh)* script for converting text files which are common to the two directories from their state in *dir1* to their state in *dir2.*

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

—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. Only recognizes the **—b** option. All other options are ignored.

—D*string*
Causes **diff** to create a merged version of *file1* and *file2* on the standard output, with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *file1,* while defining *string* will yield *file2.*

EXAMPLES

The following invocation performs the difference function on the files pgm.version1 and pgm.version2. The results are written on standard output.

```
diff pgm.version1 pgm.version2
```

FILES

/tmp/d?????	Temporary files
/usr/lib/diffh	half-hearted (**—h**) diff function

RETURN VALUE

[0]	No differences.
[1]	There are differences.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[INTERNAL]	An unexpected error occurred. Execution was terminated. Record the message and save the core file for analysis. Contact service personnel at your Tektronix field office.

CAVEATS

Editing scripts produced under the **—e** or **—f** option are naive about creating lines consisting of a single '.'.

When comparing directories with the **—b** option specified, **diff** first compares the files ala *cmp(1),* and then decides to run the **diff** algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string differences.

SEE ALSO

cmp(1), cc(1), comm(1), ed(1), diff3(1), pr(1).

NAME

diff3 — 3–way differential file comparison

SYNOPSIS

diff3 [—{**e**,**x**,**3**}] *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.

OPTIONS

The following options are mutually exclusive:

—e 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.

—x Produces a script to incorporate only changes flagged = = = =. The following command will apply the resulting script to *file1*.

```
(cat script; echo '1,$p') | ed - file1
```

—3 Same as **—x** except that it works on changes flagged = = = = 3.

EXAMPLES

The following example will print on standard out a script for **ed** that will incorporate into *ver1* all the changes flagged = = = =.

```
diff3 -x ver1 ver2 ver3
```

FILES

/tmp/d3?????	Temporary data file.
/usr/lib/diff3	Output formatter.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[INTERNAL]	An unexpected error occurred. Execution was terminated. Record the message and save the core file for analysis. Contact service personnel at your Tektronix field office.

CAVEATS

Text lines that consist of a single '.' will defeat **—e.**

SEE ALSO

diff(1), rcsdiff(1rcs).

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*. *Name1* and *name2* are the old and new versions of the file. *name2* may be given as '—', which will cause **diffmk** to read from the standard input. *name1* may not be specified as '—'. **Diffmk** generates *name3*, if it exists, which contains the lines of *name2* plus inserted formatter "change mark" (**.mc**) requests. If *name3* was not specified the result is placed on standard out. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

If anyone is so inclined, **diffmk** can be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c > diff.nr
nroff macs diff.nr
```

where the file *macs* contains:

```
.pl  1
.ll  77
.nf
.eo
.nc
```

The **.ll** request might specify a different line length, depending on the nature of the program being printed. The **.eo** and **.nc** requests are probably needed only for C programs.

If the characters | and * are inappropriate, a copy of **diffmk** can be edited to change them (*diffmk* is a shell procedure).

EXAMPLES

The following example generates a file *changes* which contains the differences between *source.nr* and *newsource.nr* marked with change bars.

```
diffmk source.nr newsource.nr diff.nr
nroff diff.nr > changes
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

The input files are expected to be *nroff(1)* input. Other types of input, including formatted text, require special macros similar to those required for C source.

Incorrect output may be generated when **diffmk** is used at the beginning of a pipe. It is therefore recommended that the output from **diffmk** be put in a file before formatting.

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.

SEE ALSO

diff(1), nroff(1).

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.

SEE ALSO

cmp(1), diff(1).

NAME

dirname — deliver directory portion of path name

SYNOPSIS

dirname *path*

DESCRIPTION

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

EXAMPLES

The following example will set the shell variable **NAME** to the name of the directory which contains the command 'sh'.

```
NAME=`dirname \`pathof sh\``
```

Note the use of **pathof** to get the full pathname of the command, and the use of escaped backquotes to allow two levels of command substitution.

CAVEATS

The **dirname** of a file name with no slashes is considered to be '.'. For example,

```
dirname foo
```

prints '.'.

SEE ALSO

basename(1), pathof(1), sh(1sh).

NAME

cd, chdir, pushd, popd, dirs — directory change commands (**csh** *built-in)*

SYNOPSIS

cd [*dirname*]
or
cd *+n*

chdir [*dirname*]
or
chdir *+n*

pushd [*dirname*]
or
pushd *+n*

popd [*+n*]

dirs

DESCRIPTION

Csh maintains the name current directory and a directory stack which can be used to keep up with all of the places the user has been and may wish to go back to. With this, the user does not have to remember where to go back to when sidetracked.

The command **cd,** and its synonym **chdir,** change the current working directory of the shell. If no argument is given, the directory is changed to the user's home directory. If *dirname* begins with '/', './', or '../', an attempt is made to change to that directory. Otherwise, the directory *dirname* is searched for in the current directory and in each element of the variable *cdpath.* If this fails, and there is a variable named the same as the value of *dirname* whose value begins with a '/', that directory is used (see EXAMPLES). When the directory is changed, the top element of the directory stack is replaced by the new directory. When the *cdpath* or directory name variable is used to change the directory, the path of the new current directory is printed. With the argument *+n,* where *n* is a number (beginning at 0), the *n*'th element of the directory stack is moved from that position to the top of the stack. The contents of the directory stack is always printed upon completion.

The command **pushd** is used to add directory names to the directory stack, and to edit the stack, as well as change the current directory. With no arguments, **pushd** exchanges the top two elements of the directory stack. With a directory name argument, the directory is changed, and the name of the new current directory is pushed on to the stack. With the argument *+n,* where *n* is a number (beginning at 0), the *n*'th element of the directory stack is moved from that position to the top of the stack. The contents of the directory stack is always printed upon completion.

The command **popd** is used to remove elements from the directory stack. With no arguments, the top element of the stack is removed, resulting in changing the current working directory. With the argument *+n,* where *n* is a number (numbers begin at 0, but '+0' is not valid), the *n*'th element of the directory stack is removed. In the latter use, the current directory is unchanged. The contents of the directory stack is always printed upon completion.

The command **dirs** prints the contents of the directory stack in order. The top element (which is the current directory) is printed first.

The variable *cdpath* contains a list of directories to search if the directory name given is not a subdirectory of the current directory. This variable is maintained along with the environment variable CDPATH, which is used by the *sh(1sh)* **cd** command, but these variables are not the same. In **csh,** the current directory is implicitly the first element in *cdpath,* whereas in **sh,** the current directory must be given explicitly. In order to cope with this difference, the value of CDPATH is imported upon startup of a new shell. CDPATH is changed when *cdpath* is changed (using *set(1csh)),* but *cdpath* is not changed when CDPATH is changed (using *setenv(1csh)).*

The variable *cwd* is set by **csh** whenever the current directory is changed. Due to symbolic links and the ditributed file system, this variable may not always contain correct or desirable data. If the value of *cwd* is required to be correct, the variable *hardpaths* may be set, which causes all changes of directory to get the current directory path by calling *getwd(3c),* which will give the correct path. This makes directory changes somewhat slower. An alternate method is to use the command *pwd(1)* to get the correct current directory path when needed.

EXAMPLES

This example shows a use of the directory name variable.

```
set default=/
set cdpath=( ~ ~/* )
cd default
```

In this case, if there is no directory named 'default' in the current directory, the user's home directory, and any subdirectories of the user's home directory, the current directory will become '/'.

The following example shows some of the features of manipulation of the directory stack. Here, the character '%' represents the **csh** prompt.

```
% dirs
~

% pushd /bin
/bin ~
```

```
% pushd /etc
/etc /bin ~
% cd /usr
% dirs
/usr /bin ~
% pushd +2
~ /usr /bin
% popd +2
~ /usr
% popd
/usr
```

VARIABLES

CDPATH	The directory change search path.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the error message occurred.

CAVEATS

Shell scripts should never change to a subdirectory and attempt to go back by executing **cd .. ** , since the directory changed to may be a symbolic link to another directory whose parent directory is different from where the last **cd** was executed. At the very least, the *cwd* variable's value should be saved and used to go back.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1sh), chdir(1sh), continue(1csh), csh(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), chdir(2).

NAME

dist — redistribute a message to additional addresses

SYNOPSIS

dist [+**folder**] [**msg**] [—**form** *formfile*] [—**editor** *editor*] [—**annotate**] [—**noannotate**] [—**inplace**] [—**noinplace**] [—**help**]

DESCRIPTION

Dist is similar to **forw**. It prepares the specified message (default is the current message) for redistribution to addresses that are not on the original address list. The file */Mail/distcomps,* or a standard form, or the file specified by the option —**form** *formfile* is used as the blank components file and prepended to the message. The standard form has the components **Resent-to:** and **Resent-cc:.** When the message is sent, the components **Resent-Date:** *date* and **Resent-From:** *name* are also prepended to the message. If you specify the —**msgid** option to **send** , the component **Resent-msgid:** is prepended to the message.

Dist uses the following components from the user's *mh_profile* file:
Path: determines the user's MH directory
Editor: overrides the use of */bin/ed* as the default editor
lasteditor—**next: defines editor to be used after exit from** *lasteditor*

OPTIONS

—**annotate**
Each message is annotated with the lines:
Distributed: date
Distributed: Resent-to: names
where each "to" list contains as many lines as required. This annotation is done only if the message is sent directly from **dist**. If the message is sent later by **send** the annotations do not take place.

—**inplace**
Annotation is done in place to preserve links to the annotated message.

—**editor** *editor*
Specifies an editor other than the default editor.

+**folder** *folder*
Folder becomes the current folder, and the current message is set to the message you are redistributing.

DEFAULTS

+**folder** defaults to the current folder
msg defaults to cur
—**editor** defaults to /bin/ed
—**noannotate**
—**noinplace**

FILES

/etc/mh/components	Default message skeleton
⟨mh-dir⟩/components	User message skeleton

$HOME/mh_profile	The user profile
⟨mh-dir⟩/draft	The default message file

CAVEATS

Only those addresses in **Resent-To: , Resent-cc: ,** and **Resent-Bcc:** are sent. Also, the folder specified in the component **Resent-Fcc:** *folder* is not honored.

Send recognizes a message as a redistribution message by the existence of the field **Resent-To: ,** so do not try to redistribute a message with only a **Resent-cc:** component.

SEE ALSO

mh(5mh).

NAME

dtoc — directory table of contents

SYNOPSIS

dtoc [*directory*]

DESCRIPTION

Output is a list of all readable subdirectories beginning at *directory*. If no *directory* is given, the list begins at the current directory.

The output is in the form of a textual table of contents readable by *vtoc(1g)*. Each line of the output represents a single directory. Each line has three fields: level number, directory name, and the number of ordinary readable files in the directory.

EXAMPLES

The following example makes a visual display on a Tektronix terminal of all the readable directories under the user's home directory.

```
dtoc $HOME ¦ vtoc ¦ td
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), erase(1g), exp(1g), find(1), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

du — summarize disk usage

SYNOPSIS

du [**—a**] [**—r**] [**—s**] [*filename ...*]

DESCRIPTION

Du gives the number of kilobytes contained in all files and (recursively) directories within each specified *filename*, if *filename* is a directory. If a *filename* argument is a regular file, the size of the file will be given only if one or more options is given. The count includes the indirect blocks of the file. If *filename* is missing, '.' is used.

Absence of an option causes an entry to be generated for each directory only.

OPTIONS

—a Causes an entry to be generated for each file.

—r Messages will be generated when directories cannot be read, files cannot be opened, etc.

—s Causes only the grand total to be given.

EXAMPLES

The following invocation gives the total disk usage for the directory */tmp*.

```
du -s /tmp
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Non–directories given as arguments (not under **—a** option) are not listed.

A file with two or more links is only counted once unless there are greater than 1024 distinctly linked files. The 1025th linked file encountered and all subsequent linked files are counted multiply.

Files with holes in them will get an incorrect block count.

SEE ALSO

df(1), lseek(2), quot(8).

NAME

ex, e, edit, vi, view — text editor

SYNOPSIS

ex [—] [—**R**] [—**l**] [—**r**] [—**t***tag*] [—**v**] [—**w***n*] [+*command*] name ...
e (**ex** options)
edit (**ex** options)
vi (**ex** options)
view (**ex** options)

DESCRIPTION

Ex is the root of a family of editors: **e, edit, ex, vi,** and **view. Ex** is a superset of **ed,** with the most notable extension being a display editing facility. Display based editing is the focus of **vi.**

If you have 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.**

Executing **vi** is the same as executing **ex** with the —**v** option.

The commands **e** and **edit** are equivalent to *ex* in all respects except that the following options are set: **nomagic** (reduced regular expression syntax), **noopen** (the commands *open* and *visual* are disabled), and **report** is set to 1 (all commands modifying more than 1 line cause a message to be printed).

The command **view** is the same as executing **ex** with the —**v** and —**R** options.

OPTIONS

— Suppresses all interactive-user feedback and is useful in processing editor scripts in command files.

—**R**
Sets the *readonly* option at the start of the editing session.

—**l** Sets up for editing LISP, setting the *showmatch* and *lisp* options.

—**r** Used in recovering after an editor or system crash, retrieving the last saved version of the named file, or if no file is specified, typing a list of saved files.

—**t***tag*
Equivalent to an initial *tag* command, editing the file containing the *tag* and positioning the editor at its definition.

—**v** Visual display editing mode.

—**w***n*
Sets the default window size to *n*.

+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 file initially. Other useful commands here are scanning patterns of the form "/pat" or line numbers, e.g. "+100" starting at line 100.

FILES

/usr/lib/ex?.?strings	Error messages.
/usr/lib/ex3.7recover	Recover command.
/usr/lib/ex3.7preserve	Preserve command.
/etc/termcap	Default file containing terminal capability entries.
~/.exrc	The editor startup file.
/tmp/Exnnnnn	Editor temporary file.
/tmp/Rxnnnnn	Named buffer temporary file.
/usr/preserve	Preservation directory.

VARIABLES

TERM	The type of terminal being used.
TERMCAP	The file containing the terminal capability entry, or the entry itself.
SHELL	The name of the shell program to be used when invoking a subshell or processing special commands.
HOME	The user's home directory.
EXINIT	Specifies an alternate editor startup file or contains the actual startup data.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

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.

If there are more than 300 magic numbers of type long and/or short in the magic number file, no files may be edited.

When metacharacters are used in naming a file (such as with the 'r' command), the user's default shell is used to do the expansion in noninteractive mode. When this shell is **csh**, the *prompt* variable should be checked to see if it is set in the *.cshrc* file. If it isn't, it should not be set, since this will cause **ex** to see more than one filename.

SEE ALSO

awk(1), e(1), ed(1), edit(1), grep(1), egrep(1), sed(1), vi(1), view(1), magic(5), termcap(5t), environ(7).

NAME

echo — echo arguments

SYNOPSIS

echo [**—n**] [*arg*] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a newline 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 up to \c without newline
\f	form–feed
\n	newline
\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.

Echo is also built into *sh* with exactly the same functionality.

OPTIONS

—n Suppress printing of the trailing newline. Equivalent to ending arguments with \c.

EXAMPLES

The following invocation of **echo** will print the sentence "This is a test.", followed by a newline.

```
echo This is a test.
```

The following invocation of **echo** will print the sentence "This is a test."; the newline will not be printed.

```
echo This is a test.\cthis text will not be printed.
```

RETURN VALUE

[NO_ERRS] Command completed without error.

CAVEATS

Beware of conflicts with the shell's use of \.

If the first argument begins with **—n** but has more characters, as in

```
echo -none
```

the argument is printed as is, and the newline is not suppressed by this argument.

The *csh* built–in command **echo** does not know about the C–like escape conventions.

Echo is actually a shell script which executes the *sh* built–in command. This is provided for programs like *make(1)* that need to execute **echo** directly.

SEE ALSO

csh(1csh), echo(1csh), echo(1sh), make(1), sh(1sh).

NAME

echo, glob — echo arguments (**csh** built-in)

SYNOPSIS

echo [**—n**] [*wordlist*]
glob [*wordlist*]

DESCRIPTION

The command **echo** is the **csh** output command. Each argument is printed, followed by a space. Unless the **—n** option is given, the arguments are followed by a newline.

The command **glob** is similar to **echo,** except that arguments are separated by null characters, there is no trailing newline, and, thus, there is no **—n** option. This is normally used by programs wishing to expand a list of words with **csh.**

OPTIONS

—n Inhibit the printing of the trailing newline.

EXAMPLES

This example prints a line that says "Current directory:" followed by the name of the current directory.

```
echo "Current directory: $cwd"
```

RETURN VALUE

[NO_ERRS] Command completed without error.

CAVEATS

The **—n** must appear alone as the first argument. Otherwise, it is ignored. This is useful if the string '-n' is to be printed.

Each argument is followed by a space, including the last one. Thus, the command "echo a b c" prints "a⟨SPACE⟩b⟨SPACE⟩c⟨SPACE⟩⟨NEWLINE⟩".

The **csh** version of **echo** does not understand escaped character sequences (like '\n'). The commands *echo(1)* and *echo(1sh)* do understand this type of sequence.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1), echo(1sh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

echo — echo arguments (*sh* built-in)

SYNOPSIS

echo [**—n**] [*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 up to \c 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.

OPTIONS

—n Suppress printing of the trailing newline. Equivalent to ending arguments with \c.

EXAMPLES

The following invocation of echo will print the sentence "This is a test.", followed by a new-line.

```
echo This is a test.
```

The following invocation of echo will print the sentence "This is a test."; the new-line will not be printed.

```
echo This is a test.\cthis text will not be printed.
```

RETURN VALUE

[NO_ERRS] Command completed without error.

CAVEATS

Beware of conflicts with the shell's use of \.

If the first argument begins with **—n** but has more characters, as in

```
echo -none
```

the argument is printed as is, and the newline is not suppressed by this argument.

For the sake of *make(1)* and other programs that execute **echo** without executing **sh,** the command is also supplied as a shell script.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1), echo(1csh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), make(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2).

NAME

ed — text editor

SYNOPSIS

ed [—] [**-p**[*prompt*] [**-q**] [**—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.

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 .IREAK).

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: non–printing characters are printed in octal, except for *backspace,* which is printed as '–', backspace, '⟨', and *tab,* which is printed as '–', backspace, '⟩'. 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*(1sh)) 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 UTek 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 .IREAK) is sent, **ed** prints a **?** and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 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 q/s1/p
?s1 ?s1?
```

OPTIONS

— 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*.

—h Turns on *help* mode. See the description of the **H** command.

—p [*prompt*]
Causes the specified *prompt* to be printed when **ed** has finished with a command and is waiting for the next one. If no *prompt* is specified, '*' is used. Without the **—p** option, no prompting is done.

—q Allows the signal SIGQUIT (normally generated by the character ^\) to terminate the edit session, and turns off the — option. Normally, SIGQUIT is ignored.

—x An *X* command is simulated first to handle an encrypted file.

FILES

/tmp/e$$	temporary; $$ is the process number.
ed.hup	work is saved here if the terminal is hung up.

DIAGNOSTICS

? for command errors.

?*filename*^
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.

VARIABLES

HOME	The user's home directory. Upon abnormal termination of **ed** , a file called *ed.hup* is placed in this directory."
SHELL	The user's login shell. Used for shell escapes.

RETURN VALUE

[NOERRS]	
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

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 *lsh*(1sh)).
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.

SEE ALSO

grep(1), red(1), sed(1), sh(1sh), stty(1).

NAME

ex, e, edit, vi, view — text editor

SYNOPSIS

ex [—] [**—R**] [**—l**] [**—r**] [**—t***tag*] [**—v**] [**—w***n*] [*+command*] name ...
e (**ex** options)
edit (**ex** options)
vi (**ex** options)
view (**ex** options)

DESCRIPTION

Ex is the root of a family of editors: **e, edit, ex, vi,** and **view. Ex** is a superset of **ed,** with the most notable extension being a display editing facility. Display based editing is the focus of **vi.**

If you have 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.**

Executing **vi** is the same as executing **ex** with the **—v** option.

The commands **e** and **edit** are equivalent to *ex* in all respects except that the following options are set: **nomagic** (reduced regular expression syntax), **noopen** (the commands *open* and *visual* are disabled), and **report** is set to 1 (all commands modifying more than 1 line cause a message to be printed).

The command **view** is the same as executing **ex** with the **—v** and **—R** options.

OPTIONS

— Suppresses all interactive–user feedback and is useful in processing editor scripts in command files.

—R
Sets the *readonly* option at the start of the editing session.

—l Sets up for editing LISP, setting the *showmatch* and *lisp* options.

—r Used in recovering after an editor or system crash, retrieving the last saved version of the named file, or if no file is specified, typing a list of saved files.

—t*tag*
Equivalent to an initial *tag* command, editing the file containing the *tag* and positioning the editor at its definition.

—v Visual display editing mode.

—w*n*
Sets the default window size to *n*.

+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 file initially. Other useful commands here are scanning patterns of the form "/pat" or line numbers, e.g. "+100" starting at line 100.

FILES

/usr/lib/ex?.?strings	Error messages.
/usr/lib/ex3.7recover	Recover command.
/usr/lib/ex3.7preserve	Preserve command.
/etc/termcap	Default file containing terminal capability entries.
~/.exrc	The editor startup file.
/tmp/Exnnnnn	Editor temporary file.
/tmp/Rxnnnnn	Named buffer temporary file.
/usr/preserve	Preservation directory.

VARIABLES

TERM	The type of terminal being used.
TERMCAP	The file containing the terminal capability entry, or the entry itself.
SHELL	The name of the shell program to be used when invoking a subshell or processing special commands.
HOME	The user's home directory.
EXINIT	Specifies an alternate editor startup file or contains the actual startup data.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

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.

If there are more than 300 magic numbers of type long and/or short in the magic number file, no files may be edited.

When metacharacters are used in naming a file (such as with the 'r' command), the user's default shell is used to do the expansion in noninteractive mode. When this shell is **csh,** the *prompt* variable should be checked to see if it is set in the *.cshrc* file. If it isn't, it should not be set, since this will cause **ex** to see more than one filename.

SEE ALSO

awk(1), e(1), ed(1), edit(1), grep(1), egrep(1), sed(1), vi(1), view(1), magic(5), termcap(5t), environ(7).

NAME

efl — Extended FORTRAN Language

SYNOPSIS

efl [options] [files]

DESCRIPTION

Efl compiles a program written in the EFL language into clean FORTRAN. **Efl** provides the same control flow constructs as does *ratfor*(1), which are essentially identical to those in C:

statement grouping with braces;
decision–making with if, if–else, and switch–case; while, for, FORTRAN do, repeat, and repeat . . . until loops; multi–level break and next. In addition, EFL has C–like data structures, and more uniform and convenient input/output syntax, generic functions. EFL also provides some syntactic features to make programs easier to read and write:

free form input:
multiple statements/line; automatic continuation statement label names (not just numbers),

comments:
this is a comment

translation of relationals:
>, >=, etc., become .GT., .GE., etc.

return (expression)
returns expression to caller from function

define:
define name replacement

include:
include filename

OPTIONS

—# Prevents comments from being copied through.

—C
Causes comments to be copied through to the FORTRAN output (default).

—w
Suppresses warning messages.

CAVEATS

If a command argument contains an embedded equal sign, that argument is treated as if it had appeared in an **option** statement at the beginning of the program. **Efl** is best used with *f77*(1).

SEE ALSO

f77(1), ratfor(1).

NAME

egrep — search a file for a pattern

SYNOPSIS

egrep [**—E**] [**—b**] [**—c**] [**—h**] [**—l**] [**—n**] [**—s**] [**—v**] [*expression*] [*filename ...*]

DESCRIPTION

Egrep searches the input *files* (standard input default) for lines matching an expression. Normally, each line found is copied to the standard output. Unless the **—h** option is given, the file name is shown if there is more than one input file.

The *expression* may be specified either on the command line or in a file. If it is given on the command line, it may be preceded by the flag **—e.** This is especially useful for specifying expressions that begin with '–'. If the *expression* is contained in a file, the name of the file must be the first file in the file list on the command line, and the **—f** option must be given.

Egrep patterns are full regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space. **Egrep** accepts extended regular expressions. In the following description 'character' excludes newline:

A \ followed by a single character other than newline matches that character.

The character ^ matches the beginning of a line.

The character **$** matches the end of a line.

A **.** (period) matches any character.

A single character not otherwise endowed with special meaning matches that character.

A string enclosed in brackets **[]** matches any single character from the string. Ranges of ASCII character codes may be abbreviated as in 'a—z0—9'. A **]** may occur only as the first character of the string. A literal — must be placed where it can't be mistaken as a range indicator.

A regular expression followed by an * (asterisk) matches a sequence of 0 or more matches of the regular expression. A regular expression followed by a + (plus) matches a sequence of 1 or more matches of the regular expression. A regular expression followed by a **?** (question mark) matches a sequence of 0 or 1 matches of the regular expression.

Two regular expressions concatenated match a match of the first followed by a match of the second.

Two regular expressions separated by ¦ or newline match either a match for the first or a match for the second.

A regular expression enclosed in parentheses matches a match for the regular expression.

The order of precedence of operators at the same parenthesis level is **[]** then ***+?** then concatenation then ¦ and newline.

Care should be taken when using the characters **$**, *****, **[**, **^**, **¦**, **(**, **)**, and **** in the *expression* as they are also meaningful to the Shell. It is safest to enclose the entire *expression* argument in single quotes.

OPTIONS

—E Print matching lines in the form :
filename, **line** *linenumber* : matching line

—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.

—c Only a count of matching lines is printed.

—e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a '—'.

—f *filename*
The regular expression is taken from the *filename.*

—h Suppress printing of file names if multiple files are given.

—l The names of files with matching lines are listed (once) separated by newlines. If this option is given when reading from standard input, **egrep** simply exits with a value of 1.

—n Each line is preceded by its relative line number in the file.

—s Silent mode. No error message is printed for nonexistent files (does not apply to regular expression file).

—v All lines but those matching are printed.

EXAMPLES

The following example prints the number of lines in the file "example" that contain either the word "and" or the word "or". The word boundaries in this case are the beginning and end of the line, and spaces or tabs (^I).

```
egrep -c '(^¦[ ^I]+)(and¦or)([ ^I]+¦$)' example
```

This example prints the lines in all of the files in the current directory which contain the sequence "number" with or without the 'n' capitalized, followed by a space and an integer. No file names are printed.

```
egrep -h '[nN]umber [0-9]+' *
```

RETURN VALUE

[0]	No errors occurred and at least one match was found.
[USAGE]	Incorrect command line syntax. Execution terminated.
[1]	No errors occurred but no matches were found.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

If one of the input files is the same as the output, as in the example "egrep re * > out", that input file is not searched in order to prevent problems. No message is printed in this case. If the old functionality is required, pipe the output through *cat(1)*.

The precedence of the output specification options is **—c, —E,** and **—l,** which turn off the **—n** and **—b** options and the printing of the file name and matching line.

Tests show that **egrep** is the fastest of the pattern searching commands.

SEE ALSO

ed(1), ex(1), fgrep(1), grep(1), regcmp(1), sed(1), sh(1sh).

NAME

erase — send erase character to Tektronix 4014 terminal

SYNOPSIS

erase

DESCRIPTION

Erase erases the screen of the Tektronix 4014 display terminal.

CAVEATS

If you are not using a 4104 terminal, **erase** may cause unexpected results.

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

error — analyze and disperse compiler error messages

SYNOPSIS

error [—l*ignorefile*] [—**S**] [—**T**] [—**n**] [—**q**] [—**s**] [—**t** *suffixlist*] [—**v**] [*filename*]

DESCRIPTION

Error analyzes and optionally disperses the diagnostic error messages produced by a number of compilers and language processors to the source file and line where the errors occurred. It can replace the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously without machinations of multiple windows in a screen editor.

Error looks at the error messages, either from the specified file *name* or from the standard input, and attempts to determine which language processor produced each error message, determines the source file and line number to which the error message refers, determines if the error message is to be ignored or not, and inserts the (possibly slightly modified) error message into the source file as a comment on the line preceding to which the line the error message refers. Error messages which can't be categorized by language processor or content are not inserted into any file, but are sent to the standard output. **Error** touches source files only after all input has been read. By specifying the **—q** query option, the user is asked to confirm any potentially dangerous (such as touching a file) or verbose action. Otherwise **error** proceeds on its merry business. If the **—t** touch option and associated suffix list is given, **error** will restrict itself to touch only those files with suffixes in the suffix list. Error also can be asked (by specifying **—v)** to invoke *vi*(1) on the files in which error messages were inserted; this obviates the need to remember the names of the files with errors.

Error is intended to be run with its standard input connected via a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into **error.** (see example).

Error knows about the error messages produced by: *make, cc, cpp, ccom, as, ld, lint, pi, pc* and *f77.* **Error** knows a standard format for error messages produced by the language processors, so is sensitive to changes in these formats. For all languages except Pascal, error messages are restricted to be on one line. Some error messages refer to more than one line in more than one files; **error** will duplicate the error message and insert it at all of the places referenced.

Error will do one of six things with error messages.

synchronize
: Some language processors produce short errors describing which file it is processing. **Error** uses these to determine the file name for languages that don't include the file name in each error message.

discard
Error messages from *lint* that refer to one of the two *lint* libraries, */usr/lib/llib-lc* and */usr/lib/llib-port* are discarded, to prevent accidently touching these libraries. Again, these error messages are consumed entirely by **error.**

nullify
Error messages from *lint* can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The names of functions to ignore are taken from either the file named *.errorrc* in the users's home directory, or from the file named by the —**l** option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.

not file specific
Error messages that can't be intuited are grouped together, and written to the standard output before any files are touched. They will not be inserted into any source file.

file specific
Error message that refer to a specific file, but to no specific line, are written to the standard output when that file is touched.

true errors
Error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are candidates for inserting into the file they refer to. Other error messages are consumed entirely by **error** or are written to the standard output. **Error** inserts the error messages into the source file on the line preceding the line the language processor found in error. Each error message is turned into a one line comment for the language, and is internally flagged with the string "###" at the beginning of the error, and "%%%" at the end of the error. This makes pattern searching for errors easier with an editor, and allows the messages to be easily removed. In addition, each error message contains the source line number for the line the message refers to. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C or Pascal, it is possible to insert a comment into another comment, which can wreak havoc with a future compilation. To avoid this, format the source program so there are no language statements on the same line as the end of a comment.

Error catches interrupt and terminate signals, and if in the insertion phase, will orderly terminate what it is doing.

OPTIONS

—l*ignorefile*
Specifies the file which contains the names of functions to ignore.

—S
Print error messages as they are processed, giving the type of error and the name of the program that produced the message (such as *cc* or *make)*

—T Terse output.

—n Do *not* touch any files; all error messages are sent to the standard output.

—q The user is *queried* whether s/he wants to touch the file. A "y" or "n" to the question is necessary to continue. Absence of the **—q** option implies that all referenced files (except those refering to discarded error messages) are to be touched.

—s Print out *statistics* regarding the error categorization. Not too useful.

—t *suffixlist*
Take the following argument as a suffix list. Files whose suffixes do not appear in the suffix list are not touched. The suffix list is dot separated, and "*" wildcards work. Thus the suffix list:

".c.y.foo*.h"

allows **error** to touch files ending with ".c", ".y", ".foo*" and ".y".

—v After all files have been touched, overlay the visual editor *vi* with it set up to edit all files touched, and positioned in the first touched file at the first error. If *vi* can't be found, try *ex* or *ed* from standard places.

EXAMPLES

The following examples put error messages resulting from compiling the files hello.c and world.c into these files at the line numbers where the errors occur, do not put errors in any header (.h) files containing errors, and put the user in the editor *vi(1)* at the place where the first error was found. The first invocation is for *sh(1sh),* and the second is for *csh(1csh).*

```
cc hello.c world.c 2>&1 ¦ error -t ".c" -v
cc hello.c world.c ¦& error -t ".c" -v
```

Note that the standard output and standard error are being redirected.

FILES

$HOME/.errorrc	File containing function names to ignore for **lint** error messages.
/dev/tty	The user's teletype.

CAVEATS

Opens the teletype directly to do user querying.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's format of error messages may cause **error** to not understand the error message.

Error, since it is purely mechanical, will not filter out subsequent errors caused by 'floodgating' initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected (error puts them before). The alignment of the ' ¦' marking the point of error is also disturbed by **error.**

Error was designed for work on CRT's at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.

SEE ALSO

as(1), cc(1), csh(1csh), ed(1), ex(1), f77(1), ld(1), lint(1), pc(1), sh(1sh).

NAME

eval — evaluate and execute arguments (**csh** built-in)

SYNOPSIS

eval [*arg...*]

DESCRIPTION

The **eval** command causes the arguments to be read by the shell as input. This is usually used to execute text generated by other commands or stored in variables.

EXAMPLES

The following example shows a standard use of the utility *tset(1),* which gets the terminal type and can generate environment variable assignments.

```
set noglob; eval `tset -s`; unset noglob
```

In this use, the **tset** command might generate output like:

```
set noglob ;
setenv TERM aaa ;
setenv TERMCAP (termcap string) ;
unset noglob
```

The **eval** command causes these commands to be run in the current shell, thus setting the TERM and TERMCAP variables. It should be noted that the extra "set noglob" in the example is required in order to prevent filename substitution in the **tset** output, and that the extra "unset noglob" in the example is recommended in case the **tset** command is aborted.

RETURN VALUE

The value returned by **eval** is that of the commands executed.

CAVEATS

A backquoted command that generates more than one word can not be executed by **csh** unless the **eval** command is used. This is used to inhibit usages like '*', which generates an ambiguous command line.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1sh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

eval — execute arguments (*sh* built-in)

SYNOPSIS

eval [*args* . . .]

DESCRIPTION

eval 's arguments are read as input to the shell and the resulting command(s) are executed. This command is useful for executing the output of commands and the contents of shell variables.

EXAMPLES

This example shows a shell script, called *pathdir,* which prints the word 'cd' followed by the name of the directory containing the name of the given executable file.

```
#!/bin/sh
if test $# -ne 1
then
        echo "$0 : usage : "`basename $0`" file" 1>&2
        exit 1
fi
where=`pathof $1`
if test $? -ne 0
then
        exit 1
fi
echo -n "cd "
expr "$where" : '^\(.*)/.*$'
```

Executing the command **eval** *pathdir sh* will effect the command *cd /bin,* since that is where the program *sh* resides.

For another example of using **eval** to execute the output from a command, see the documentation for *tset(1).*

Another use of **eval** is to store commands in shell variables and have them execute. For example, the commands:

```
w='who|sort'
eval $w
```

will cause the command 'who ¦ sort' to be executed. Note that simply typing '$w' on the command line will not work in this case. The reason for this is that the shell has already passed the stage of evaluating the '¦' when it expands the variable.

It is possible to have shell variables which execute correctly without **eval** as in the following example:

```
w=`eval who|sort`
$w
```

It is important to note that the commands are run in the current shell, not a subshell. This means that the following use of a shell variable works as expected:

```
bin=`eval cd /bin;pwd`
$bin
```

The command **$bin** will print the line '/bin' and the current shell becomes /bin.

RETURN VALUE

Eval returns the exit status of the last command executed.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1csh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2).

NAME

ex, e, edit, vi, view — text editor

SYNOPSIS

ex [—] [**—R**] [**—l**] [**—r**] [**—t***tag*] [**—v**] [**—w***n*] [*+command*] name ...
e (**ex** options)
edit (**ex** options)
vi (**ex** options)
view (**ex** options)

DESCRIPTION

Ex is the root of a family of editors: **e, edit, ex, vi,** and **view. Ex** is a superset of **ed,** with the most notable extension being a display editing facility. Display based editing is the focus of **vi.**

If you have 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.**

Executing **vi** is the same as executing **ex** with the **—v** option.

The commands **e** and **edit** are equivalent to **ex** in all respects except that the following options are set: **nomagic** (reduced regular expression syntax), **noopen** (the commands *open* and *visual* are disabled), and **report** is set to 1 (all commands modifying more than 1 line cause a message to be printed).

The command **view** is the same as executing **ex** with the **—v** and **—R** options.

OPTIONS

— Suppresses all interactive-user feedback and is useful in processing editor scripts in command files.

—R
Sets the *readonly* option at the start of the editing session.

—l Sets up for editing LISP, setting the *showmatch* and *lisp* options.

—r Used in recovering after an editor or system crash, retrieving the last saved version of the named file, or if no file is specified, typing a list of saved files.

—t*tag*
Equivalent to an initial *tag* command, editing the file containing the *tag* and positioning the editor at its definition.

—v Visual display editing mode.

—w*n*
Sets the default window size to *n*.

+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 file initially. Other useful commands here are scanning patterns of the form "/pat" or line numbers, e.g. "+100" starting at line 100.

FILES

/usr/lib/ex3.7recover	Recover command.
/usr/lib/ex3.7preserve	Preserve command.
/etc/termcap	Default file containing terminal capability entries.
$HOME/.exrc	The editor startup file.
/tmp/Exnnnnn	Editor temporary file.
/tmp/Rxnnnnn	Named buffer temporary file.
/usr/preserve	Preservation directory.

VARIABLES

TERM	The type of terminal being used.
TERMCAP	The file containing the terminal capability entry, or the entry itself.
SHELL	The name of the shell program to be used when invoking a subshell or processing special commands.
HOME	The user's home directory.
EXINIT	Specifies an alternate editor startup file or contains the actual startup data.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

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.

If there are more than 300 magic numbers of type long and/or short in the magic number file, no files may be edited.

When metacharacters are used in naming a file (such as with the 'r' command), the user's default shell is used to do the expansion in noninteractive mode. When this shell is **csh**, the *prompt* variable should be checked to see if it is set in the *.cshrc* file. If it isn't, it should not be set, since this will cause **ex** to see more than one filename.

SEE ALSO

awk(1), e(1), ed(1), edit(1), grep(1), egrep(1), sed(1), vi(1), view(1), magic(5), termcap(5t), environ(7).

NAME

exec — execute command in place of the current shell (**csh** built-in)

SYNOPSIS

exec *command* [*arg...*]

DESCRIPTION

The **exec** command executes the given command in place of the current shell. Since there are a limited number of processes available for each user, it is useful to be able to execute the last command in place of the shell so that the process isn't hanging around with nothing to do.

The *command* may not be a **csh** built-in.

EXAMPLES

This example shows an interface to the command *make(1),* which can be used by someone with a lot of commands and directories to keep up with. The environment variable MAKEDB contains the name of a file which contains entries of the form

```
name⟨TAB⟩directory⟨TAB⟩description
```

where the *name* is the name given to the script, the *directory* is the pathname of the directory where **make** should be run, and *description* is a message to be printed before executing the **make** command.

```
#!/bin/csh -f
set Myname=`basename $0`
#
# Check usage.
#
if ($#argv != 1) then
        echo "$Myname : usage : $Myname target"
        exit 1
endif
#
# This awk script looks through the database for the entry.
#
eval `awk -Ft '{ if ($1 == name) { printf "set Dir=%s ;
echo %s", $2, $3; exit; } }' name="$1" "$MAKEDB" `

#
# Change to the named directory and run the command.
#

cd "$Dir"
exec make "$1"
```

Note: the *awk(1)* script must be given on a single line. It is split here for readability.

Another use of **exec** is to exit the login shell without executing commands from the *.logout* file. For example, executing the command "exec clear" in the login shell will clear the screen and log the user out.

VARIABLES

PATH The user's execution search path.

RETURN VALUE

The return value is the return value of the command executed, or 1 if the command does not exist.

CAVEATS

The **exec** command never returns, even if the command was not found.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1sh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), execve(2), fork(2).

NAME

exec — executes arguments without creating a new process (*sh* built-in)

SYNOPSIS

exec [*arg* . . .]

DESCRIPTION

The command specified by *arg*, 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.

The builtin *sh* command **login** is equivalent to the command **exec login ...**

EXAMPLES

The system has a limit on the number of processes that can exist at any time and a limit on the number of active processes for each user (typically 20). The average user will very rarely reach this limit, but sometimes a number of processes will get caught in infinite loops or run for a long time. When this happens, the user can not kill any of the processes because one process slot is required to execute **kill.**

In this situation, the command

```
exec kill -9 0
```

will kill all processes currently running. Of course, this means that the user must log in again, but as stated before, this happens very rarely.

The following shell script fragment uses **exec** with no arguments to internally redirect the standard output.

```
#!/bin/sh
if test \( $# -ge 2 \) -a \( "x$1" = "x-f" \)
then
        exec > "$2"
        shift
        shift
fi
other commands ...
```

RETURN VALUE

Exec returns the exit status of the command executed. If the command does not exist, the exit code will be **NO_CMD.**

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1csh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2).

These synchronization messages are consumed entirely by **error.**

NAME

exit — causes shell to exit (*sh* built–in)

SYNOPSIS

exit [*n*]

DESCRIPTION

exit causes a shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. An end–of–file will also exit from the shell. The shell variable $? will contain the exit status of any process.

EXAMPLES

The following shell script exits with the number of arguments it is given.

```
#!/bin/sh
exit $#
```

RETURN VALUE

The exit status of **exit** is the value of the argument, or the status of the last command executed.

CAVEATS

The default exit status comes from the exit status of the last command executed. This includes built–in commands such as **break** and **continue**, which always exit with 0.

Functions are executed in the current shell, so they should use **return** instead of **exit** unless they are used to terminate the shell.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1csh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2), ERROR(3c), exit(3c).

NAME

exp — exponential function

SYNOPSIS

exp [**—c***n*] [*vector ...*]

DESCRIPTION

Output is a vector with elements e raised to the x power, where e is 2.71828, approximately, and x are the elements from the input *vector(s)*. If no *vector* is given, the standard input is assumed.

OPTIONS

—c*n*

n is the number of output elements per line.

EXAMPLES

The following example outputs the exponential of each element of A, three per line.

```
exp -c3 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

expand — expand tabs to spaces

SYNOPSIS

expand [*—tabstop,[tabstop . . .]*] [*filename* . . .]

DESCRIPTION

Expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. **Expand** is useful for pre–processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single *tabstop* argument is given then tabs are set *tabstop* spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

The tabstop numbers must be between 1 and 256 and given in increasing order.

EXAMPLES

The following example expands the tabs in the file 'text' at columns 4, 15, 34, 72, and 84 and puts the output in the file 'text.t'.

```
expand -4,15,34,72,84 text > text.t
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

unexpand(1).

NAME

export — exports variables to the environment (*sh* built–in)

SYNOPSIS

export [*filename* . . .]

DESCRIPTION

The 'environment' consists of a list of names and corresponding values which contain information that the user wants all processes to know about. When a file is executed, a copy of the environment is given to the process. In *sh,* the names in the environment are copied into shell variables for use in shell scripts. In order to change the values of the entries in the environment, the **export** command must be used.

The given names are marked for automatic export to the environment of subsequently executed commands. If no arguments are given, a list of names marked for export is printed. Note that this command is not a verb, meaning that the variable is *marked* for export, not put in the environment immediately.

If the **—a** option is set, all variables are marked for export when set or changed.

Variables which have not been marked for export are not copied into the environment, which means that the current value will not be known by any child processes. There is no way for a child to change the environment of its parent.

Shell functions may not be exported.

There is an alternate way of changing the environment for a single command. If the command name is preceded by variable assignments, these values are copied into the environment for the execution of the command. For example, the line

```
TERM=aaa printenv TERM
```

will cause the word 'aaa' to be printed, no matter what TERM was set to before. The value of the variable is not changed in the current shell.

OPTIONS

–a Marks all variables for export when set or changed.

EXAMPLES

Certain variables, such as TERM and TERMCAP, are set during execution of the *.profile* file. These must be exported so that programs such as *ex* and *more* may use them. The following portion of

the *.profile* file sets the TERM and TERMCAP variables and exports them for use by other programs.

```
TERM=aaa-30-s
TERMCAP=/etc/termcap
export TERM TERMCAP
```

RETURN VALUE

[NO_ERRS] Command completed without error.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2), getenv(3c), environ(7).

NAME

expr — evaluate arguments as an expression

SYNOPSIS

expr arg ...

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument. Expressions are expected to be integers and strings. Floating point numbers are not supported.

The operators and keywords are listed below. The list is in order of increasing precedence, with equal precedence operators grouped.

expression **|** *expression*
yields the first *expression* if it is neither null nor '0', otherwise yields the second *expression.*

expression **&** *expression*
yields the first *expression* if neither *expression* is null or '0', otherwise yields '0'.

expression relop expression
where *relop is one of* < <= = != >= > , yields '1' if the indicated comparison is true, '0' if false. The comparison is numeric if both *expression* are integers, otherwise lexicographic.

expression + *expression*
expression — *expression*
addition or subtraction of the arguments.

expression * *expression*
expression / *expression*
expression **%** *expression*
multiplication, division, or remainder of the arguments.

expression : *expression*
The matching operator compares the string first argument with the regular expression second argument; regular expression syntax is the same as that of *ed*(1). The \(...\) pattern symbols can be used to select a portion of the first argument. Otherwise, the matching operator yields the number of characters matched ('0' on failure).

match *expression expression*
This is a synonym for the : operator.

substr *string start length*
The substring operator yields the portion of *string* beginning at position *start* that is a maximum of *length* characters long. *start* must be an expression which yields a positive integer, and *length* must be an expression which yields a non–negative integer.

length *expression*
The length operator yields the length of the string *expression.*

index *string character-list*
The index operator yields the position of the first occurrence of any character in *character-list* found in *string.*

(*expression* **)**
parentheses for grouping.

EXAMPLES

To add 1 to the Shell variable *a* (where *a* is already set) :

```
a=`expr $a + 1`
```

To find the filename part (least significant part) of the pathname stored in variable *a,* which may or may not contain '/' (this is partially equivalent to *basename(1)*):

```
expr $a : '.*/\(.*\)' '|' $a
```

Note the quoted Shell metacharacters.

This example prints the portion of the home directory name following the first '/' character after the first character in the name. (Note the escaped /, which would normally mean division.):

```
expr substr \
        \( substr $HOME 2 1024 \) \
        \( index \( substr $HOME 2 1024 \) '\/' \) + 1 \
        1024
```

If $HOME is */usr/guest* , the above expression would print 'guest'.

RETURN VALUE

[0]	The expression is neither null nor '0'.
[1]	The expression is either null or '0'.
[2]	The expression contains errors.

CAVEATS

The strings *substr, index, length,* and *match* may appear only as operators, and not as strings. Therefore, an expression like 'substr : [ubs]*' results in a syntax error.

SEE ALSO

basename(1), sh(1sh), test(1sh).

NAME

f77 — FORTRAN 77 compiler

SYNOPSIS

f77 [option] ... *filename* ...

DESCRIPTION

F77 is the UTek FORTRAN 77 compiler. It accepts several types of arguments:

Arguments whose names end with '.f' are taken to be FORTRAN 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with '.o' substituted for '.f'.

Arguments whose names end with '.F' are also taken to be FORTRAN 77 source programs; these are first processed by the C preprocessor before being compiled by **f77.**

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

Arguments whose names end with '.r' or '.e' are taken to be Ratfor or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by **f77**.

OPTIONS

The following options have the same meaning as in *cc*(1). See *ld*(1) for load-time options.

—c Suppress loading (do not produce 'a.out' file) and produce '.o' files for each source file.

—go
Have the compiler produce additional symbol table information for *sdb*(1). Also pass the **—lg** flag to *ld*(1).

—o output
Name the final output file *output* instead of 'a.out'.

—p Prepare object files for profiling, see *prof*(1).

—pg
Causes the compiler to produce counting code in the manner of **—p,** but invokes a run-time recording mechanism that keeps more extensive statistics and produces a *gmon.out* file at normal termination. An execution profile can then be generated using *gprof*(1).

—w
Suppress all warning messages. If the option is '—w66', only FORTRAN 66 compatibility warnings are suppressed.

—D*name* = *def*

—D*name*
Define the *name* to the C preprocessor, as if by '#define'. If no definition is given, the name is defined as "1". (This option can be used with '.F' suffix files only.).

—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 directories on a standard list. (This option can be used with '.F' suffix files only.).

—O
Invoke an object–code optimizer.

—S
Compile the named programs, and leave the assembler–language output on corresponding files suffixed '.s'. (No '.o' file is created.).

The following options are peculiar to **f77**.

—i2
On machines which support short integers, make the default integer constants and variables short. (**—i4** is the standard value of this option). All logical quantities will also be short.

—m
Apply the *M4* preprocessor to each '.r' file before transforming it with the *Ratfor* or *EFL* preprocessor.

—onetrip
Compile DO loops that are performed at least once if reached. (FORTRAN 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

—u Make the default type of a variable 'undefined' rather than using the default FORTRAN rules.

—v Print the version number of the compiler, and the name of each pass as it executes.

—C
Compile code to check that subscripts are within declared array bounds.

—F Apply the C, *EFL*, or *Ratfor* preprocessors to relevant files, put the result in the file with the suffix changed to '.f', but do not compile.

—E*x*
Use the string *x* as an *EFL* option in processing '.e' files.

—R*x*
Use the string *x* as a *Ratfor* option in processing '.r' files.

—N [**qxscn**]*nnn*
Make static tables in the compiler bigger. The compiler will complain if it overflows its tables and suggest you apply one or more of these flags. These flags have the following meanings:

q Maximum number of equivalenced variables. Default is 150.

x Maximum number of external names (common block names, subroutine and function names). Default is 200.

s Maximum number of statement numbers. Default is 401.

c Maximum depth of nesting for control statements (e.g. DO loops). Default is 20.

n Maximum number of identifiers. Default is 1009.

—U
Do not convert upper case letters to lower case. The default is to convert FORTRAN programs to lower case except within character string constants.

—d Print debugging information. Gives RCS id information on all pieces of the compiler and tells you when and with what arguments it (the pieces of the compiler) is called with. The information given by this option should be included with any bug reports on the FORTRAN compiler.

Other arguments are taken to be either loader option arguments, or **f77**–compatible object programs, typically produced by an earlier run, or perhaps libraries of **f77**–compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name 'a.out'.

FILES

file.[fFresc]	input file
file.o	object file
a.out	loaded output
tmp/fort[pid]	temporary
/usr/lib/f77pass1	f77 pass 1 compiler
/lib/f1	pass 2
/lib/c2	optional optimizer
/lib/cpp	C preprocessor
/usr/lib/libF77.a	f77 intrinsic function library
/usr/lib/libI77.a	FORTRAN I/O library
/usr/lib/libU77.a	f77 UTek system interface library
/usr/lib/libF77_p.a	profiling f77 intrinsic function library
/usr/lib/libI77_p.a	profiling FORTRAN I/O library

/usr/lib/libU77_p.a	profiling UTek system interface library
/lib/libc.a	C library, see section 3
mon.out	file produced for analysis by *prof(1)*.
gmon.out	file produced for analysis by *gprof(1)*.

DIAGNOSTICS

The diagnostics produced by **f77** itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

CAVEATS

The optimizer occasionally makes mistakes; it should be avoided when debugging if apparently incorrect results are obtained.

SEE ALSO

cc(1), efl(1), gprof(1), ld(1), m4(1), prof(1), ratfor(1).

NAME

true, false — provide truth values

SYNOPSIS

true

false

DESCRIPTION

True and **false** are usually used in a Bourne shell script. They return the appropriate status "true" or "false" and are usually used in "while" and "until" loops.

These commands are also built–in *sh* commands for speed.

EXAMPLES

```
while true
do
command list
done
```

RETURN VALUE

[0]	The equivalent of true in **sh** conditional tests.
[1]	The equivalent of false in **sh** conditional tests.

SEE ALSO

csh(1csh), sh(1sh), true(1sh).

NAME

true, false — provide truth values (**sh** built–in)

SYNOPSIS

true

false

DESCRIPTION

True and **false** return appropriate status "true" or "false" for use in shell scripts. They are usually used in "while" and "until" loops.

These commands are also available as regular commands for use in other shells and programs.

EXAMPLES

```
while true
do
command list
done
```

RETURN VALUE

[0]	The equivalent of true in **sh** conditional tests.
[1]	The equivalent of false in **sh** conditional tests.

SEE ALSO

csh(1csh), sh(1sh), true(1).

NAME

fg, bg — foreground and background jobs (**csh** *built-in)*

SYNOPSIS

fg [**%***job*...]
or
%*job*
bg [**%***job*...]
or
%*job***&**

DESCRIPTION

The command **fg** brings a stopped or backgrounded job into the foreground, meaning that the job becomes attached to the terminal. The command **bg** puts a stopped job in the background, meaning that the job becomes detached from the terminal. The syntax for *job* is described in the manual page *jobs(1csh).* If no job argument is given, the current job is used (this job is marked by a '+' in the listing given by **jobs .)**

The syntax **%***job* is an alternate for **fg,** and **%***job***&** is an alternate for **bg.**

EXAMPLES

A common way to follow a *make(1)* command's progress is to redirect the output to a file and use the command **tail –f***file* to view the output. Usually, the *tostop* mode of the terminal is used (see *stty(1)), so that the command stops* is ready. To do this, the command

```
tail –f make_output &
```

is executed. When it is ready to send output to the terminal, a message like:

```
[2] + Stopped (tty output) tail –f make_output
```

is printed. At this point, any of the following commands can be used to bring the command into the foreground for viewing of the output:

```
fg
fg %2
fg %tail
%2
%tail
```

When the output has stopped or slowed to the point that it would be best to wait for more output to be built up, the character ^Z (control–z) can be typed to stop the command. At this point, the command can be put back into the background by using one of the following commands:

```
bg
bg %2
bg %tail
%2 &
%tail &
```

DIAGNOSTICS

command: No such job.
The job named (either implicitly or explicitly) is not a job.

command: Ambiguous.
There is more than one job that matches the pattern given.

CAVEATS

When *tostop* mode is set and a lot of jobs are running, it can be very confusing, since the job that was last announced as being the current job may no longer be so.

Backgrounding an already backgrounded job will cause the job to be marked so that it does not become the current job again automatically.

SEE ALSO

@(1csh), alias(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), signal(3c).

NAME

fgrep — search a file for a pattern

SYNOPSIS

fgrep [**—E**] [**—b**] [**—c**] [**—h**] [**—i**] [**—l**] [**—n**] [**—s**] [**—v**] [**—x**] [**—y**] *string-list* [*filename ...*]

DESCRIPTION

Fgrep searches the input *files* (standard input default) for lines containing or matching strings. Normally, each line found is copied to the standard output. Unless the **—h** option is given, the filename is shown if there is more than one input file.

The *string-list* may be either specified on the command line, or contained in a file. If the *string-list* is specified on the command line, the strings are separated by newlines, and the **—e** option may precede the string list, which is useful for string lists whose first string begins with a '–'. If the *string-list* is contained in a file, the filename must appear as the first argument after the options, and the option list must contain the option **—f.** In either case, the *string-list* must come after all other options.

OPTIONS

—E Print matching lines in the format:
filename, **line** *linenumber* : matching line

—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.

—c Only a count of matching lines is printed.

—e *string-list*
Same as a simple *string-list* argument, but useful when the *string-list* begins with a —.

—f *file*
The string list is taken from the *file.*

—h Suppress printing of filenames even if more than one filename is given.

—i, —y
Ignore case of characters specified as lower case.

—l The names of files with matching lines are listed (once) separated by newlines. If this option is given when reading from standard input, **fgrep** simply exits with a value of 1.

—n Each line is preceded by its relative line number in the file.

—s Silent mode. No error messages are printed for nonexistent files (does not apply to the string list file).

—v All lines but those matching are printed.

—x (Exact) only lines matched in their entirety are printed.

EXAMPLES

The following example searches the all of the files in the current directory for the sequence "date" and prints the names of the files that contain this sequence.

```
fgrep -l date *
```

This can be used to copy all of the files that contain the sequence "date" to the standard output with the following example.

```
cat `fgrep -l date *`
```

This example prints the lines in the file "example" containing any of the words in the file "words".

```
fgrep -f words example
```

RETURN VALUE

[0]	No errors occurred and at least one match was found.
[USAGE]	Incorrect command line syntax. Execution terminated.
[1]	No errors occurred but no matches were found.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

If one of the input files is the same as the output, as in the example "egrep re * > out", that input file is not searched in order to prevent problems. No message is printed in this case. If the old functionality is required, pipe the output through *cat(1)*.

The maximum number of strings that **fgrep** can search for varies for different size strings.

The precedence of the output specification options is **—c**, **—E**, and **—l**, which turn off the **—n** and **—b** options and the printing of the file name and matching line.

Tests show that *egrep* is the fastest of the pattern searching commands.

SEE ALSO

cat(1), egrep(1), grep(1), regcmp(1), sh(1sh).

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.

Each file is printed in the format:

name: file type Symbolic links are printed in the format:

name: Symbolic link. File pointed to is ⟨file type⟩

File uses the file */usr/lib/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. The file format is described in the manual page *magic(5)*.

File recognizes the following file types:

ASCII archives	Empty
ASCII text	English text
Assembler program text	FORTRAN progra
BASIC program text	MH message
Block special	News article
C program text	*Nroff, troff, tbl,* o
Character special	Old archive
Commands text	Old executable
Compacted text	Packed text
Cpio archives	Program scripts
Data	RCS file
Demand paged pure executable	Read only executa
Directories	Troff output

If the file is executable and not stripped, that fact is noted.

There are two ASCII archive formats: normal and long. In each case, if the archive contains a symbol definition table (see *ar(1)* and *ranlib(1)),* that fact is noted.

A program script file is a file whose first line is an interpreter line (see *execve(2)).* The name of the program to be executed is printed.

If an ASC.B file contains characters with the eighth bit set, the file type is followed by the words "with garbage".

OPTIONS

—c Causes **file** to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **—c**.

—**f***ffile*
The next argument is taken to be a file containing the names of the files to be examined.

—**m***mfile*
Instructs **file** to use an alternate magic file.

EXAMPLES

The following example will print the name of each file in */da/tmp* followed by its classification.

```
file /da/tmp/*
```

RETURN VALUE

[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NO_ERRS]	Command completed without error.

CAVEATS

It often makes mistakes. In particular it often suggests that command files are C programs.

Does not recognize Pascal or LISP.

Interpreter lines ('#! *pathname)* are not checked for validity. A line like '#! /foo' will cause the file to be identified as script commands for /foo, even though the file will not be executed. Execute permission is not checked.

SEE ALSO

ld(1), notmagic(3c), a.out(5), ar(5), cpio(5), magic(5), rcsfile(5rcs).

NAME

find — find files

SYNOPSIS

find pathname–list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each pathname in the *pathname-list* (i.e., one or more pathnames) 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*.

The primaries may be combined using the following operators (in order of decreasing precedence):

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).

2) The negation of a primary ('!' is the unary *not* operator).

3) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries). Note that the primaries **—print, —follow, —cpio,** and **—exec** are always true and always perform their operations when they are evaluated.

4) Alternation of primaries ('**—o**' is the *or* operator).

OPTIONS

—atime *n*
True if the file has been accessed in *n* days.

—cpio *name*
Write the current file to the file *name* in *cpio(5)* format (5120 byte records).

—exec *command*
True if the executed command returns a zero value as exit status. The end of the command must be punctuated by a space followed by an escaped semicolon. A command argument '{}' is replaced by the current pathname.

—follow *type*
Always true. This option determines whether symbolic links to directories will be followed. By default, symbolic links to directories are followed except when the directory exists on a different host from the top directory being searched (see EXAMPLES). The *type* may be **h** or **r**. The **h** type says that symbolic links to directories are not to be followed (in other words, follow only "hard" links). The **r** type says that remote directories (via the DFS) are to be followed as well. Note that if both **h** and **r** are given, the **r** is ignored, since there is no way to follow a remote directory without following symbolic links.

—group *gname*
True if the file belongs to group *gname* (group name or numeric group ID).

—inum *n*
True if the file has inode number *n*.

—links *n*
True if the file has *n* links.

—mtime *n*
True if the file has been modified in *n* days.

—name *filename*
True if the *filename* argument matches the current filename. Normal Shell argument syntax may be used if escaped (watch out for '[', '?' and '*').

—newer *file*
True if the current file has been modified more recently than the argument *file*.

—ok *command*
Like **—exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response **y**.

—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*.

—print
Always true; causes the current pathname to be printed.

—size *n*
True if the file is *n* blocks long (512 bytes per block).

—type *c*
True if the type of the file is *c*, where *c* is **b, c, d, f** or **l** for block special file, character special file, directory, plain file, or symbolic link. Unless the search starts from '/', a symbolic link to a file has both type **l** and the type of the file pointed to by the symbolic link, except in the case of symbolic links to nonexistent files.

—user *uname*
True if the file belongs to the user *uname* (login name or numeric user ID).

EXAMPLES

To remove all files named 'a.out' or '*.o' that have not been accessed for a week:

```
find / \( -name a.out -o -name '*.o' \)
-atime +7 -exec rm {} \;
```

The following examples print the names of all files found in the directory '//stuff/usr/bin' (note that this is a DFS pathname). Assume that the file '//stuff/usr/bin/other' is a symbolic link to the directory '//otherhost/bin' (a DFS reference to a different host). The first command below will print all filenames in '//stuff/usr/bin' and all subdirectories, including the filenames in '//stuff/usr/bin/other'. The second command will not print the filenames in '//stuff/usr/bin/other'.

```
find //stuff/usr/bin -print -follow r
find //stuff/usr/bin -print
```

FILES

/etc/passwd	System user information
/etc/group	System group information

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Note that the semicolon following the **-exec** or **-ok** aruments must be a separate argument.

Symbolic links to directories are treated just like normal directories, unless the search starts from '/'. In order to do this, information about each directory is stored.

No directory is ever searched more than once.

There is no way to selectively follow symbolic links to directories.

The **—cpio** option does not know about symbolic links. When a symbolic link is encountered, the file pointed to by the link is archived.

SEE ALSO

cpio(1), du(1), sh(1sh), test(1sh), cpio(5), fs(5).

NAME

finger, f — user information lookup program

SYNOPSIS

finger [**—b**] [**—f**] [**—h**] [**—i**] [**—l**] [**—m**] [**—p**] [**—q**] [**—s**] [**—w**] [*username ...*]

DESCRIPTION

By default **finger** lists the login name, full name, terminal name and write status (as a '*' before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each user currently logged in. (Idle time is minutes if it is a single integer, hours and minutes if a ':' is present, or days and hours if a 'd' is present.)

A longer format also exists and is used by **finger** whenever a list of peoples names is given. (Account names as well as first and last names of users are accepted.) This format is multi-line, and includes all the information described above as well as the user's home directory and login shell, any plan which the person has placed in the file *.plan* in their home directory, and the project on which they are working from the file *.project* also in the home directory.

A quick format also exists, which prints the username, login port, and login time.

Finger gets personal information about a user from the *gecos* field in the entry for the user in the file */etc/passwd*. This field is manipulated by the user via the **chfn(1)** command. The format of this field is determined by the file */usr/lib/chfn*. This file contains the labels used for the table headings of the reports printed by **finger.** If there is no */usr/lib/chfn* **finger** uses a set of default labels for the output headers. These default labels are as follows:

Name	This is the users full name.
Extension	This is the users office phone number.
Mail Stop	This is the users company mail address.
Home Phone	This is the users home phone number.
Machine	This is the name of the machine where the users primary activity is.

If */usr/lib/chfn* exists, its contents will be used instead of the default labels. It can be changed to suit the needs of the user. (See *chfn(1)* and *chfn(5)).* **Finger's** output will change accordingly.

OPTIONS

—b Brief version of long format. Suppress printing of login shell and home directory.

—f Suppress printing of headers on listings.

—h Suppress printing of the *.project* file.

—i Print data in quick format with idle time.

—l Force long output format.

—m
Match arguments only on user name.

—p Suppress printing of the *.plan* file.

—q Print data in quick format.

—s Force short output format.

—w Print data in narrow short format, which doesn't give the full name or titles for other gecos field elements.

FILES

/etc/utmp	Login data
/etc/passwd	User account data file
/usr/adm/lastlog	Last login times
$HOME/.plan	Personal plan file
$HOME/.project	Personal project file

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Only the first line of the *.project* file is printed.

SEE ALSO

chfn(1), w(1), who(1n), chfn(5).

NAME

floor — floor function

SYNOPSIS

floor [**—c***n*] [*vector ...*]

DESCRIPTION

Output is a vector with each element being the largest integer less than the corresponding element from the input *vector(s)*. If no *vector* is given, the standard input is assumed.

OPTIONS

—c*n*

n is the number of output elements per line.

EXAMPLES

The following example outputs the floor of each element of A, three per line.

```
floor -c3 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

fmt — format after concatenation

SYNOPSIS

fmt [**—i**] [**—m**] *filename ...*

DESCRIPTION

Fmt normally reads each *file* in sequence and writes it on the standard output after performing a simple formatting operation. Thus

```
fmt file
```

prints the file and

```
fmt file1 file2 >file3
```

concatenates the first two files, formats them, and places the result on the third.

If no *filename* is given **fmt** reads from the standard input.

The *comp(1mh)* and *repl(1mh)* programs in the MH mail system call fmt in response to the format response to "What now?". This invocation uses both the **—i** and **—m** switches.

OPTIONS

—i Causes each file to be formatted individually and the result placed back into the file.

—m

The input is an MH style mail message and the mail header should be left untouched.

CAVEATS

Beware of 'fmt a b >a' and 'fmt a b >b', which destroy input files before reading them.

SEE ALSO

cat(1), comp(1mh), cp(1), nroff(1), pr(1), repl(1mh).

NAME

folder — set/list current folder/message

SYNOPSIS

folder [+**folder**] [**msg**] [—**pack**] [—**nopack**] [—**all**] [—**fast**] [—**nofast**] [—**up**] [—**down**] [—**header**] [—**noheader**] [—**total**] [—**nototal**] [—**help**]

folders <equivalent to 'folder —all'>

DESCRIPTION

Since the MH environment is the shell, it is easy to lose track of the current folder from day to day. **Folder** will list the current folder, the number of messages in it, the range of the messages (low–high), and the current message within the folder, and will flag a selection list or extra files if they exist. An example of the output is:

inbox+ has 16 messages (3— 22); cur= 5.

If a +*folder* and/or *msg* are specified, they will become the current folder and/or message. An **—all** switch will produce a line for each folder in the user's MH directory, sorted alphabetically. These folders are preceded by the read–only folders, which occur as mh_profile 'cur—' entries. For example,

```
Folder # of messages ( range ); cur msg (other files)
Folder # of messages ( range ); cur msg (other files)
/fsd/rs/m/tacc has 35 messages ( 1- 35); cur= 23.
/rnd/phyl/Mail/EP has 82 messages ( 1-108); cur= 82.
notes has 2 messages ( 1- 2); cur= 1.
inbox+ has 16 messages ( 3- 22); cur= 5.
ff has 4 messages ( 1- 4); cur= 1.
mh has 76 messages ( 1- 76); cur= 70.
misc has 1 message ( 1- 1).
junk has no messages.
ucom has 124 messages ( 1-124); cur= 6; (select).

TOTAL = 340 messages in 9 Folders
```

The '+' after inbox indicates that it is the current folder. The '(select)' indicates that the folder ucom has a selection list produced by **pick**. If 'others' had appeared in parentheses at the right of a line, it would indicate that there are files in the folder directory that don't belong under the MH file naming scheme.

The header is output if either an **—all** or a **—header** switch is specified; it is suppressed by **—noheader**, Also, if **folder** is invoked by a name ending with 's' (e.g., **folders**), **—all** is assumed. A **—total** switch will produce only the summary line. If **—fast** is given, only the folder name (or names in the case of **—all**) will be listed (This is faster because the folders need not be read).

The switches **—up** and **—down** change the folder to be the one above or below the current folder. That is, **folder**—*down* will set the folder to *<current—folder>/select*, and if the current folder is a selection–list folder,

folder—*up* will set the current folder to the parent of the selection-list (See **pick** for details on selection-lists).

The switch **—pack** causes the messages in the selected folder(s) to be renumbered. This eliminates the holes in the sequence caused by removing messages. It does not change the current message number or any of the deleted messages (comma files—see rmm).

If *+folder* and/or *msg* are given, they will become the current folder and/or message.

folder has the following defaults:

+folder	defaults to the current folder
msg	defaults to none
—nofast	
—noheader	
—nototal	

Your *.mh_profile* can contain the following entries:

Path: To determine the user's MH directory
Current-Folder: To find the default current folder

OPTIONS

—all
Produces a list of all the folders in the MH directory, sorted alphabetically.

—down
Changes the current folder to the one next in the folder list.

—fast
Prints only the names of the folder or folders. (This is faster because the folders do not have to be read.)

—header
Prints the folder header line.

—help
Prints a list of the possible switches for **folder**.

—nofast
Suppresses the **fast** function.

—noheader
Suppresses the **header** function.

—nopack
Suppresses the **pack** function.

—nototal
Suppresses the **total** function.

—pack
Renumbers the messages in a folder, eliminating holes in the numbering sequence.

—total
Prints the summary line only.

—up
Changes the current folder to the previous one in the folder list.

FILES

$HOME/.mh_profile	The user profile
/bin/ls	To fast–list the folders

SEE ALSO

comp(1mh), fmt(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), mh(1mh), mhl(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh), mh(5mh), mh_profile(5mh).

NAME

folder — set/list current folder/message

SYNOPSIS

folder [+folder] [msg] [—pack] [—nopack] [—all] [—fast] [—nofast] [—up] [—down] [—header] [—noheader] [—total] [—nototal] [—help]

folders <equivalent to 'folder —all'>

DESCRIPTION

Since the MH environment is the shell, it is easy to lose track of the current folder from day to day. **Folder** will list the current folder, the number of messages in it, the range of the messages (low–high), and the current message within the folder, and will flag a selection list or extra files if they exist. An example of the output is:

inbox+ has 16 messages (3— 22); cur= 5.

If a *+folder* and/or *msg* are specified, they will become the current folder and/or message. An **—all** switch will produce a line for each folder in the user's MH directory, sorted alphabetically. These folders are preceded by the read–only folders, which occur as mh_profile 'cur—' entries. For example,

```
Folder # of messages ( range ); cur msg (other files)
Folder # of messages ( range ); cur msg (other files)
/fsd/rs/m/tacc has 35 messages ( 1- 35); cur= 23.
/rnd/phyl/Mail/EP has 82 messages ( 1-108); cur= 82.
notes has 2 messages ( 1- 2); cur= 1.
inbox+ has 16 messages ( 3- 22); cur= 5.
ff has 4 messages ( 1- 4); cur= 1.
mh has 76 messages ( 1- 76); cur= 70.
misc has 1 message ( 1- 1).
junk has no messages.
ucom has 124 messages ( 1-124); cur= 6; (select).

TOTAL = 340 messages in 9 Folders
```

The '+' after inbox indicates that it is the current folder. The '(select)' indicates that the folder ucom has a selection list produced by **pick**. If 'others' had appeared in parentheses at the right of a line, it would indicate that there are files in the folder directory that don't belong under the MH file naming scheme.

The header is output if either an **—all** or a **—header** switch is specified; it is suppressed by **—noheader**, Also, if **folder** is invoked by a name ending with 's' (e.g., **folders**), **—all** is assumed. A **—total** switch will produce only the summary line. If **—fast** is given, only the folder name (or names in the case of **—all**) will be listed (This is faster because the folders need not be read).

The switches **—up** and **—down** change the folder to be the one above or below the current folder. That is, **folder***—down* will set the folder to *<current—folder>/select*, and if the current folder is a selection–list folder,

folder—*up* will set the current folder to the parent of the selection–list (See **pick** for details on selection–lists).

The switch **—pack** causes the messages in the selected folder(s) to be renumbered. This eliminates the holes in the sequence caused by removing messages. It does not change the current message number or any of the deleted messages (comma files—see rmm).

If *+folder* and/or *msg* are given, they will become the current folder and/or message.

folder has the following defaults:

+folder	defaults to the current folder
msg	defaults to none
—nofast	
—noheader	
—nototal	

Your *.mh_profile* can contain the following entries:

Path: To determine the user's MH directory
Current–Folder: To find the default current folder

OPTIONS

—all
Produces a list of all the folders in the MH directory, sorted alphabetically.

—down
Changes the current folder to the one next in the folder list.

—fast
Prints only the names of the folder or folders. (This is faster because the folders do not have to be read.)

—header
Prints the folder header line.

—help
Prints a list of the possible switches for **folder**.

—nofast
Suppresses the **fast** function.

—noheader
Suppresses the **header** function.

—nopack
Suppresses the **pack** function.

—nototal
Suppresses the **total** function.

—pack
Renumbers the messages in a folder, eliminating holes in the numbering sequence.

—total
Prints the summary line only.

—up
Changes the current folder to the previous one in the folder list.

FILES

$HOME/.mh_profile	The user profile
/bin/ls	To fast-list the folders

SEE ALSO

comp(1mh), fmt(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), mh(1mh), mhl(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh), mh(5mh), mh_profile(5mh).

NAME

forw — forward messages

SYNOPSIS

forw [+**folder**] [*msgs*] [—**editor** *editor*] [—**form** *formfile*]
[—**annotate**] [—**noannotate**] [—**inplace**] [—**noinplace**]
[—**wait**] [—**nowait**] [—**output**] [—**nooutput**] [—**build**]
[—**help**]

DESCRIPTION

Forw prepares a mail message containing other messages. **Forw** constructs the new message from the *components* file or from *formfile* (specified with the —**form** option) with a body composed of the message(s) to be forwarded. An editor is invoked (as described in *comp(1mh)*), and after editing is complete, you are prompted before the message is sent.

If you specify the —**annotate** option, each message being forwarded is annotated with the lines:

Forwarded: date
Forwarded: To: names
Forwarded: cc: names

Each 'To:' and 'cc:' list contains as many lines as required to list *names*. This annotation is done only if you send the message directly from **forw**. If you don't send the forwarded message, you can use **comp —use** in a later session to re-edit and send the message, but the annotations won't take place.

Your *.mh_profile* file may contain the following entries:

Path: Your MH mail directory
Editor: Overrides the default editor, */bin/prompter*
Current-Folder: The default current folder
⟨lasteditor⟩-next: The editor used after exit from ⟨lasteditor⟩

Forw has the following defaults:

+*folder* defaults to the current folder
msgs defaults to cur (the current message)
—**editor** defaults to /bin/prompter
—**noannotate**
—**noinplace**
—**wait**
—**output**

OPTIONS

—**annotate**
Each message being forwarded is annotated with the lines

Forwarded: date
Forwarded: To: names
Forwarded: cc: names

—editor *editor*
Use *editor* instead of the default text editor. See *comp(1mh)*.

—form *formfile*
Use *formfile* (from your MH directory) as the skeleton format.

—help
Display a synopsis of the **forw** command.

—inplace
Annotate a message in place to preserve its links.

—noannotate
Don't add annotations to forwarded messages.

—noinplace
Do not annotate a message in place.

+folder
Folder becomes the current folder, and the first message being forwarded becomes the current message.

—wait
Wait until the message is sent before terminating.

—nowait
Terminate immediately after selecting send at the *What now?* prompt. This is for sending messages quickly.

—output
Print error messages to the terminal.

—nooutput
Don't print error messages.

—build
Build the forwarded message but don't send it. You can send the message later with **comp —use** (see *comp(1mh)*).

FILES

/usr/lib/mh/components	The message skeleton
⟨mh-dir⟩/components	Used instead of the standard skeleton
$HOME/.mh_profile	Your MH mail profile file
⟨mh-dir⟩/draft	The default message file

SEE ALSO

comp(1mh), fmt(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), mh(1mh), mhl(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh), mh(5mh), mh_profile(5mh).

NAME

fpr — print FORTRAN file

SYNOPSIS

fpr

DESCRIPTION

Fpr is a filter that transforms files formatted according to FORTRAN's carriage control conventions into files formatted according to UTek line printer conventions.

Fpr copies its input onto its output, replacing the carriage control characters with characters that will produce the intended effects when printed using *lpr*(1mdqs). The first character of each line determines the vertical spacing as follows:

Character	Vertical Space Before Printing
Blank	One line
0	Two lines
1	To first line of next page
+	No advance

A blank line is treated as if its first character is a blank. A blank that appears as a carriage control character is deleted. A zero is changed to a newline. A one is changed to a form feed. The effects of a "+" are simulated using backspaces.

EXAMPLES

a.out ¦ fpr ¦ lpr

fpr < f77.output ¦ lpr

CAVEATS

Results are undefined for input lines longer than 170 characters.

SEE ALSO

f77(1).

NAME

fsplit — split a multi-routine FORTRAN file into individual files

SYNOPSIS

fsplit [**-e** efile] ... [file]

DESCRIPTION

Fsplit takes as input either a file or standard input containing FORTRAN source code. It attempts to split the input into separate routine files of the form *name.f,* where *name* is the name of the program unit (e.g. function, subroutine, block data or program). The name for unnamed block data subprograms has the form *blkdtaNNN.f* where NNN is three digits and a file of this name does not already exist. For unnamed main programs the name has the form *mainNNN.f.* If there is an error in classifying a program unit, or if *name.f* already exists, the program unit will be put in a file of the form *zzzNNN.f* where *zzzNNN.f* does not already exist.

OPTIONS

—*e* Normally each subprogram unit is split into a separate file. When the option is used, only the specified subprogram units are split into separate files.

EXAMPLES

The following example will split readit and doit into separate files.

```
fsplit -e readit -e doit prog.f
```

DIAGNOSTICS

If names specified via the *-e* option are not found, a diagnostic is written to *standard error.*

CAVEATS

Fsplit assumes the subprogram name is on the first noncomment line of the subprogram unit. Nonstandard source formats may confuse **fsplit.**

It is hard to use *-e* for unnamed main programs and block data subprograms since you must predict the created file name.

SEE ALSO

csplit(1).

NAME

ftp — file transfer program

SYNOPSIS

ftp [**—v**] [**—d**] [**—i**] [**—n**] [**—g**] [**—p***port*] [**host**]

DESCRIPTION

Ftp is the user interface to the ARPANET standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which **ftp** is to communicate may be specified on the command line. If this is done, **ftp** will immediately attempt to establish a connection to an FTP server on that host; otherwise, **ftp** will enter its command interpreter and await instructions from the user. When **ftp** is awaiting commands from the user the prompt "ftp〉" is provided the user. The following commands are recognized by **ftp**:

! Invoke a shell on the local machine.

append *local-file* [*remote-file*]
Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

ascii Set the file transfer *type* to network ASCII. This is the default type.

bell Arrange that a bell be sounded after each file transfer command is completed.

binary Set the file transfer *type* to support binary image transfer.

bye Terminate the FTP session with the remote server and exit **ftp**.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

close Terminate the FTP session with the remote server, and return to the command interpreter.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified it is used to set the debugging level. When debugging is on, **ftp** prints each command sent to the remote machine, preceded by the string "—〉".

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote

machine is used. If no local file is specified, output comes to the terminal.

form *format*
Set the file transfer *form* to *format*. The default format is "file".

get *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine. The current settings for *type*, *form*, *mode*, and *structure* are used while transferring the file.

hash Toggle hash-sign ("#") printing for each data block transferred. The size of a data block is 1024 bytes.

glob Toggle file name globbing. With file name globbing enabled, each local file or pathname is processed for *csh*(1csh) metacharacters. These characters include "*?[]~{}". Remote files specified in mutliple item commands, e.g. *mput*, are globbed by the remote server. With globbing disabled all files and pathnames are treated literally.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, **ftp** prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, the output is sent to the terminal.

mdelete *remote-files*
Delete the specified files on the remote machine. If globbing is enabled, the specification of remote files will first be expanded using *ls*.

mdir *remote-files local-file*
Obtain a directory listing of multiple files on the remote machine and place the result in *local-file*.

mget *remote-files*
Retrieve the specified files from the remote machine and place them in the current local directory. If globbing is enabled, the specification of remote files will first be expanding using *ls*.

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Obtain an abbreviated listing of multiple files on the remote machine and place the result in *local-file*.

mode [*mode-name*]
Set the file transfer *mode* to *mode-name*. The default mode is "stream" mode.

mput *local-files*
Transfer multiple local files from the current local directory to the current working directory on the remote machine.

open *host* [*port*]
Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, **ftp** will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), **ftp** will also attempt to automatically log the user in to the FTP server (see below).

prompt Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default), any *mget* or *mput* will transfer all files.

put *local-file* [*remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for bye.

quote *arg1 arg2 ...*
The arguments specified are sent, verbatim, to the remote FTP server. A single FTP reply code is expected in return.

recv *remote-file* [*local-file*]
A synonym for get.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

remotestatus [*path-name*]
Request status of the remote FTP server. If path–name is given then remote server will return status of the remote path–name.

rename [*from*] [*to*]
Rename the file *from* on the remote machine, to the file *to*.

rmdir *directory-name*
Delete a directory on the remote machine.

send *local-file* [*remote-file*]
A synonym for put.

sendport
Toggle the use of PORT commands. By default, **ftp** will attempt to use a PORT command when establishing a connection for each data transfer. If the PORT command fails, **ftp** will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they've been accepted.

status Show the current status of **ftp**.

struct [*struct-name*]
Set the file transfer *structure* to *struct-name*. By default "stream" structure is used.

tenex Set the file transfer type to that needed to talk to TENEX machines.

trace Toggle packet tracing.

type [*type-name*]
Set the file transfer *type* to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, **ftp** will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. Unless **ftp** is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

verbose
Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]
A synonym for help.

Command arguments which have embedded spaces may be quoted with quote (") marks.

FILE NAMING CONVENTIONS

Files specified as arguments to **ftp** commands are processed according to the following rules.

1) If the file name "—" is specified, the **stdin** (for reading) or **stdout** (for writing) is used.

2) If the first character of the file name is "¦", the remainder of the argument is interpreted as a shell command. **Ftp** then forks a

shell, using *popen*(3s) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. ''"¦ ls –lt"''. A particularly useful example of this mechanism is: ''dir ¦more''.

3) Failing the above checks, if ''globbing'' is enabled, local file names are expanded according to the rules used in the *csh*(1csh); c.f. the *glob* command.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The *type* may be one of ''ascii'', ''image'' (binary), ''ebcdic'', and ''local byte size'' (for PDP–10's and PDP–20's mostly). **Ftp** supports the ascii and image types of file transfer at this time.

Ftp supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

RESPONSES

In verbose mode all responses from the remote hoste are printed, otherwise only error responses are printed. Responses are prefixed by one char indicating the nature of the response:

› Positive Prelimnary reply; "Okay so far."

* Positive Completion reply; "Okay."

\+ Positive Intermediate reply: "Okay, expect more."

! Transient Negative reply; "No, I have problems (try again)."

? Permanent Negative reply: "No, that is an error"

INTERRUPTING

A transfer in progress can be changed by interrupting **Ftp.** The actions possible after an interrupt are:

abort
abort the transfer.

status
print out status of local and remote sides.

quit leave ftp program.

continue
resume file transfer.

OPTIONS

Options may be specified at the command line, or to the command interpreter.

—v (verbose on) option forces **ftp** to show all responses from the remote server, as well as report on data transfer statistics.

—n option restrains **ftp** from attempting "auto–login" upon initial connection. If auto–login is enabled, **ftp** will check the *.netrc* file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, **ftp** will use the login name on the local machine as the user identity on the remote machine, and prompt for a password and, optionally, an account with which to login.

—i option turns off interactive prompting during mutliple file transfers.

—d option enables debugging.

—g option disables file name globbing.

—p*num* use the tcp port *num* instead of the port listed for **ftp/tcp** in **/etc/services**.

RETURN VALUE

[NO_ERRS] Command completed without error.

[USAGE] Incorrect command line syntax. Execution terminated.

[NP_ERRS]

CAVEATS

Many FTP server implementations do not support the experimental operations such as print working directory.

Some implementations do not support the interrupting of transfers for aborting or status. In this case, the user will have to reconnect with **open** command.

SEE ALSO

netrc(5n), ftpd(8n).

NAME

gamma — gamma function

SYNOPSIS

gamma [**—c***n*] [*vector ...*]

DESCRIPTION

Output is the gamma value for each element of the input *vector(s)*. If no *vector* is given, the standard input is assumed.

OPTIONS

—c*n*

n is the number of output elements per line.

EXAMPLES

The following example outputs the gamma value for each element of A, three per line.

```
gamma -c3 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

gas — generate additive sequence

SYNOPSIS

gas [—c*n*] [—i*n*] [—n*n*] [—s*n*] [—t*n*]

DESCRIPTION

Output is a vector of *number* elements determined by the parameters *start*, *terminate*, and *interval*, and by the formula

$$x[i] = start + (i * interval \text{ MOD } (terminate - start + interval))$$

The parameters *number*, *start*, *terminate*, and *interval* are set by command options.

OPTIONS

—c*n*

n elements per output line.

—i*n*

interval: = *n*. If not given, *interval*: = 1.

—n*n*

number: = *n*. If not given, *number*: = 10, unless *terminate* is given, then *number*: = ¦*terminate–start*¦/*interval*.

—s*n*

start: = *n*. If not given, *start*: = 1.

—t*n*

terminate: = *n*. If not given, *terminate*: = positive infinity. The default value of *number* usually terminates generation before positive infinity is reached.

EXAMPLES

The following example generates the numbers 1 to 10

```
gas
```

The following example generates .01,.02,.03,.04,.05

```
gas -s.01,t.05,i.01
```

The following example generates 3,5,3,5

```
gas -s3,t5,i2,n4
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

gcore — get core images of running processes

SYNOPSIS

gcore *process-id ...*

DESCRIPTION

Gcore creates a core image of each specified process, suitable for use with *adb(1)* or *sdb(1)*.

FILES

/dev/cvt	table of kernel symbols
/dev/kmem	image of kernel memory
/dev/drum	image of swap space
core.⟨process-id⟩	core images

DIAGNOSTICS

Can't read kernel symbols
: */dev/cvt*, the table of kernel symbols, could not be accessed. (See *cvt*(4)).

Process not found
: A core image for a process was not created because the process could not be found in the process table.

Not owner
: A core image for a process was not created because the caller is not the owner of the process.

System process
: A core image for a process was not created because the process is a system process.

Process exiting
: A core image for a process was not created because the process is exiting.

Zombie
: A core image for a process was not created because the process is a "zombie" process.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Paging activity that occurs while **gcore** is running may cause the program to become confused. For best results, the desired processes should be stopped.

SEE ALSO

adb(1), sdb(1), cvt(4).

NAME

gd — GPS dump

SYNOPSIS

gd [*file* ...]

DESCRIPTION

Gd prints a human readable listing of a Graphical Primitive String (GPS) file. If no *file* is given the standard input is used.

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

ged — graphical editor

SYNOPSIS

ged [**—euRr***n*] [*GPS file* ...]

DESCRIPTION

Ged is an interactive graphical editor used to display, construct, and edit GPS files on Tektronix 4010 series and 4100 series display terminals. To use color, hardware text and markers, and to erase selectively on the 4100 series terminal, the *$TERM* environment variable must be set appropriately. If GPS *file*(s) are given, **ged** reads them into an internal display buffer and displays the buffer. The GPS in the buffer can then be edited. If — is given as a file name, **ged** reads a GPS from the standard input.

A GPS file is composed of instances of three types of graphical objects: *lines*, *arc*, and *text*. *Arc* and *lines* objects have a start point, or *object-handle*, followed by zero or more points, or *point-handles*. *Text* has only an object–handle. The objects are positioned within a Cartesian plane, or *universe*, having 64K (—32K to +32K) points, or *universe-units*, on each axis. The universe is divided into 25 equal sized areas called *regions*. Regions are arranged in five rows of five squares each, numbered 1 to 25 from the lower left of the universe to the upper right.

Ged maps rectangular areas, called *windows*, from the universe onto the display screen. Windows allow the user to view pictures from different locations and at different magnifications. The *universe-window* is the window with minimum magnification, i.e., the window that views the entire universe. The *home-window* is the window that completely displays the contents of the display buffer.

COMMANDS

Ged commands are entered in *stages*. Typically you end each stage by pressing the ⟨Return⟩ key. Prior to the final ⟨Return⟩ the command may be aborted by pressing ⟨Rubout⟩ (or your *interrupt* key, if different). The input of a stage may be edited during the stage using your *erase* and *kill* keys. The prompt * indicates that **ged** is waiting at stage 1.

Each command consists of a subset of the following stages:

1. *Command line*
   A *command line* consists of a command *name* followed by *argument*(s) followed by a ⟨Return⟩. A command *name* is a single character. Command *arguments* are either *option*(s) or a *file-name*. *Options* are indicated by a leading —.

2. *Text* *Text* is a sequence of characters terminated by an unescaped ⟨Return⟩. (120 lines of text maximum.)

3. *Points* *Points* is a sequence of one or more screen locations (maximum of 30) indicated either by the terminal crosshairs or by name. The prompt for entering *points* is the appearance of the crosshairs. When the crosshairs are visible, typing:

   ⟨**Space**⟩
   (space bar) enters the current location as a *point*. The *point* is identified with a number.

   $*n* enters the previous *point* numbered *n*.

   ⟩*x* labels the last *point* entered with the upper case letter *x*.

   $*x* enters the *point* labeled *x*.

   . establishes the previous *points* as the current *points*. At the start of a command the previous *points* are those locations given with the previous command.

   = echoes the current *points*.

   $.*n* enters the *point* numbered *n* from the previous *points*.

   # erases the last *point* entered.

   @ erases all of the *points* entered.

4. *Pivot* The *pivot* is a single location, entered by typing ⟨Return⟩ or by using the **$** operator, and indicated with a *.

5. *Destination*
   The *destination* is a single location entered by typing ⟨Return⟩ or by using **$**.

COMMAND SUMMARY

In the summary, characters typed by the user are printed in **bold**. Command stages are printed in *italics*. Arguments surrounded by brackets "**[]**" are optional. Parentheses "**()**" surrounding arguments separated by "or" means that exactly one of the arguments must be given.

Construct commands:

Arc	[—**c**olor,**e**cho,**s**tyle,**w**eight] *points*
Box	[—**c**olor,**e**cho,**s**tyle,**w**eight] *points*
Circle	[—**c**olor,**e**cho,**s**tyle,**w**eight] *points*
Hardware	[—**e**cho] *text points*
Lines	[—**c**olor,**e**cho,**s**tyle,**w**eight] *points*
Text	[—**a**ngle,**c**olor,**e**cho,**h**eight,**m**id–point,right–point,**t**ext,**w**eight] *text points*

Edit commands:

Delete	(— (**u**niverse or **v**iew) or *points*)
Edit	[—**a**ngle,**c**olor,**e**cho,**h**eight,**s**tyle,**w**eight] (— (**u**niverse or **v**iew) or *points*)
Kopy	[—**e**cho,**p**oints,**x**] *points pivot destination*
Move	[—**e**cho,**p**oints,**x**] *points pivot destination*
Rotate	[—**a**ngle,**e**cho,**k**opy,**x**] *points pivot destination*
Scale	[—**e**cho,**f**actor,**k**opy,**x**] *points pivot destination*

View commands:

coordinates	*points*
erase	
new–display	
object–handles	(— (**u**niverse or **v**iew) or *points*)
point–handles	(— (**l**abelled–points or **u**niverse or **v**iew) or *points*)
view	(— (**h**ome or **u**niverse or **r**egion) or [—**x**] *pivot destination*)
x	[—**v**iew] *points*
zoom	[—**o**ut] *points*

Other commands:

quit or **Q**uit	
read	[—**a**ngle,**e**cho,**h**eight,**m**id–point,**r**ight–point,**t**ext,**w**eight *file-name* [*destination*]
set	[—**a**ngle,**c**olor,**e**cho,**f**actor,**g**raphtext,**h**eight,**k**opy,**m**id–point,**p**oints, **r**ight–point,**s**tyle,**t**ext,**w**eight,**x**]
write	*file-name*
!command	
?	

Options:

Options specify parameters used to construct, edit, and view graphical objects. If a parameter used by a command is not specifed as an *option*, the default value for the parameter will be used (see **s**et below). The format of command *options* is:

—*option* [,*option*]

where *option* is *keyletter*[*value*]. Flags take on the *values* of true or false indicated by + and — respectively. If no *value* is given with a flag, true is assumed.

Object options:

angle*n*	Angle of *n* degrees.
color*c*	Color (4100 series only) is *c*, where *c* may be a color index ($0 \leq c < 8$), or *w*hite, *r*ed, *g*reen, or *b*lue, assuming the factory default color map.
echo	When true, echo additions to the display buffer.
factor*n*	**S**cale factor is *n* percent.
graphtext	When true, use hardware text (4100 series only).
height*n*	Height of *text* is *n* universe-units ($0 \leq n < 1280$).
kopy	When true, copy rather than move.

mid–point When true, mid–point is used to locate text string.

points When true, operate on points otherwise operate on objects.

right–point When true, right–point is used to locate *text* string.

style*type* Line style set to one of following *types*:

so	solid
da	dashed
dd	dot–dashed
do	dotted
ld	long–dashed

text When false, *text* strings are outlined rather than drawn.

weight*type* Sets line weight to one of following *types*:

n	narrow
m	medium
b	bold

Area options:

home Reference the home–window.

out Reduce magnification.

region*n* Reference region *n*.

universe Reference the universe–window.

view Reference those objects currently in view.

x Indicate the center of the referenced area.

COMMAND DESCRIPTIONS

Construct commands:

Arc and **L**ines
behave similarly. Each consists of a *command line* followed by *points*. The first *point* entered is the object–handle. Successive *points* are point–handles. **L**ines connect the handles in numerical order. **A**rc fits a curve to the handles (currently a maximum of 3 points will be fit with a circular arc; splines will be added in a later version).

Box and **C**ircle
are special cases of **L**ines and **A**rc, respectively. **B**ox generates a rectangle with sides parallel to the universe axes. A diagonal of the rectangle would connect the first *point* entered with the last *point*. The first *point* is the object–handle. Point–handles are created at each of the vertices. **C**ircle generates a circular arc centered about the *point* numbered zero and passing through the last *point*. The circle's object–handle coincides with the last *point*. A point–handle is generated 180 degrees around the circle from the object–handle.

Text and **H**ardware
generate *text* objects. Each consists of a *command line*, *text* and *points*. *Text* is a sequence of characters delimited by ⟨Return⟩. Multiple lines of text may be entered by preceding a ⟨Return⟩ with a backslash (i.e., \⟨Return⟩). The **T**ext command creates software generated characters. Each line of software text is treated as a separate *text* object. The first *point* entered is the object–handle for the first line of text. The **H**ardware command sends the characters in *text* uninterpreted to the terminal.

Edit commands:

Edit commands operate on portions of the display buffer called *defined areas*. A defined area is referenced either with an area *option* or interactively. If an area *option* is not given, the perimeter of the defined area is indicated by *points*. If no *point* is entered, a small defined area is built around the location of the ⟨Return⟩. This is useful to reference a single *point*. If only one *point* is entered, the location of the ⟨Return⟩ is taken in conjunction with the *point* to indicate a diagonal of a rectangle. A defined area referenced by *points* will be outlined with dotted lines.

Delete
removes all objects whose object–handle lies within a defined area. The **u**niverse option removes all objects and erases the screen.

Edit modifies the parameters of the objects within a defined area. Parameters that can be edited are:

angle	angle of *text*
color	color of *lines*, *arc*, and *text*.
height	height of *text*
style	style of *lines* and *arc*
weight	weight of *lines*, *arc*, and *text*.

Kopy (or **M**ove)
copies (or moves) object– and/or point–handles within a defined area by the displacement from the *pivot* to the *destination*.

Rotate
rotates objects within a defined area around the *pivot*. If the **k**opy flag is true then the objects are copied rather than moved.

Scale
For objects whose object handles are within a defined area, point displacements from the *pivot* are scaled by **f**actor percent. If the **k**opy flag is true then the objects are copied rather than moved.

View commands:

coordinates
prints the location of *point*(s) in universe– and screen–units.

erase
clears the screen (but not the display buffer).

new–display
erases the screen then displays the display buffer.

object–handles (or **p**oint–handles)
labels object–handles (and/or point–handles) that lie within the defined area with **O** (or **P**). **P**oint–handles identifies labeled points when the labelled–points flag is true.

view moves the window so that the universe point corresponding to the *pivot* coincides with the screen point corresponding to the *destination*. Options for **h**ome, **u**niverse, and **r**egion display particular windows in the universe.

x indicates the center of a defined area. Option **v**iew indicates the center of the screen.

zoom
decreases (**z**oom **o**ut) or increases the magnification of the viewing window based on the defined area. For increased magnification, the window is set to circumscribe the defined area. For a decrease in magnification the current window is inscribed within the defined area.

Other commands:

quit or **Q**uit
exit from **ged**. **q**uit responds with **?** if the display buffer has not been written since the last modification.

read inputs the contents of a file. If the file contains a GPS it is read directly. If the file contains text it is converted into *text* object(s). The first line of a text file begins at *destination*.

set when given *option*(s) resets default parameters, otherwise it prints current default values.

write
outputs the contents of the display buffer to a file.

! escapes **ged** to execute a UTek system command.

? lists **ged** commands.

OPTIONS

Ged accepts the following command–line options:

—e Do not erase the screen before the initial display.

—r*n*
Display region number *n*.

—u Display the entire GPS *universe*.

—R
Restricted shell invoked on use of **!**.

VARIABLES

TERM The user's terminal type.

CAVEATS

See Appendix A of the *Tektronix 4014 Computer Display Terminal User's Manual* for the proper terminal strap options.

SEE ALSO

The Graphics Editor in *UTek Tools, Volume 2.*

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g)* in the *UTek Command Reference.*

NAME

get — get a version of an SCCS file

SYNOPSIS

get [—r*SID*] [—**c***cutoff*] [—**i***list*] [—**x***list*] [—**a***seq-no.*]

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. 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.

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).

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*(1sccs)) 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.

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*(1sccs)), 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*(1sccs)).

%F%
SCCS file name.

%P%
Fully qualified SCCS file name.

%Q%
The value of the **q** flag in the file (see *admin*(1sccs)).

%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*(1sccs).

%W%
A shorthand notation for constructing *what*(1sccs) strings for UTek program files. %W% = ~%Z%%M%⟨horizontal–tab⟩%I%

%A%
Another shorthand notation for constructing *what*(1sccs) strings for non–UTek program files. %A% = ~%Z%%Y%~%M%~%I%%Z%

OPTIONS

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*(1sccs) 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"**.

—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*(1sccs). 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*(1sccs)). 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*(1sccs)) 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*(1sccs)) 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 | .ft 2) — .ft 2)

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 ".ft 2) 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 (5sccs)*). This keyletter is used by the *comb*(1sccs) 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.

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*(1sccs)) 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.

DIAGNOSTICS

Use *sccshelp*(1sccs) for explanations.

CAVEATS

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.

SEE ALSO

admin(1sccs), delta(1sccs), sccshelp(1sccs), prs(1sccs), what(1sccs), sccsfile(5sccs).

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.

EXAMPLES

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
```

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

SEE ALSO

sh(1sh), getopt(3c).

NAME

echo, glob — echo arguments (**csh** built-in)

SYNOPSIS

echo [**—n**] [*wordlist*]
glob [*wordlist*]

DESCRIPTION

The command **echo** is the **csh** output command. Each argument is printed, followed by a space. Unless the **—n** option is given, the arguments are followed by a newline.

The command **glob** is similar to **echo,** except that arguments are separated by null characters, there is no trailing newline, and, thus, there is no **—n** option. This is normally used by programs wishing to expand a list of words with **csh.**

OPTIONS

—n Inhibit the printing of the trailing newline.

EXAMPLES

This example prints a line that says "Current directory:" followed by the name of the current directory.

```
echo "Current directory: $cwd"
```

RETURN VALUE

[NO_ERRS] Command completed without error.

CAVEATS

The **—n** must appear alone as the first argument. Otherwise, it is ignored. This is useful if the string '–n' is to be printed.

Each argument is followed by a space, including the last one. Thus, the command "echo a b c" prints
"a<SPACE>b<SPACE>c<SPACE><NEWLINE>".

The **csh** version of **echo** does not understand escaped character sequences (like '\n'). The commands *echo(1)* and *echo(1sh)* do understand this type of sequence.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1), echo(1sh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

goto — jump to labelled line (**csh** built–in)

SYNOPSIS

goto *word*

DESCRIPTION

The specified *word* is filename and command expanded to yield a *label*. The shell searches for a line of the form *label*:, rewinding the input as far as possible, or prompting for input if the label has not been defined. Execution continues after the labelled line.

EXAMPLES

The following shell script looks through each of the files given. If any of the files does not contain a "comment" line (a line that begins with a '#'), the script prints a message and aborts.

```
#!/bin/csh -f
set OK=no
foreach File ($argv)
        grep -l '^#' "$File" >& /dev/null
        if ($status != 0) goto abort
        cat "$File"
        set OK=yes
end
if (OK == "yes") exit
abort:
        echo "Command aborted : $File contains no comment lines."
        exit 1
```

CAVEATS

The functionality of this command can be obtained by using *continue(1csh)* and *break(1csh)* without the trouble of seeking through the input.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), break(1sh), cd(1csh), chdir(1csh), continue(1csh), continue(1sh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), grep(1), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

gpconf — display or change GPIB device configuration

SYNOPSIS

gpconf —a
gpconf gpib*n*
gpconf *instrument*
gpconf *device* [**default**] [**slot** *slot*] [**addr** *pri*[*.sec*]] [**eom** *byte*] [**—eom**] [**htime** *secs*] [**ptime** *secs*] [[—]**dma**] [[—]**poll**] [[—]**sc** [[—]**std1**] [[—]**tcs**] [[—]**vstd1**] [[—]**weoi**]
gpconf *device* **reset**
gpconf gpib*n* **slot** *slot*

DESCRIPTION

Gpconf displays or changes the configuration of GPIB devices. The first three forms display the current configuration of all GPIB devices (**—a**), a GPIB interface and attached instruments (**gpib***n*), or a single instrument. The fourth form adds or changes the configuration of the named device and prints the new configuration. The fifth form resets the configuration to the stored settings. The sixth form is used to move all the instruments attached to one interface to a different slot. The new device is configured to the the old settings; the old device is removed from the configuration.

OPTIONS

default — Change to default configuration. The default settings for an *interface* device (**gpib***n*) are:

slot 0 addr 0 htime 5 ptime 0.1 eom EOI dma sc tcs —std1 —vstd1.

The default settings for an instrument device are:

slot 0 htime 5 ptime 0.1 eom EOI dma poll.

Note that there is no default address for instruments; the address must always be specified explicitly.

slot *slot* — Specifies the interface slot to which the instrument is attached. The built–in interface is slot 0.

addr *pri*[*.sec*] — Specifies the GPIB address for the interface or instrument. You may not specify a secondary address for an interface.

eom *byte* — Specifies a byte to be recognized as end–of–message. *Byte* may be **EOI**, **CR** (carriage return, hex 0D), **LF** (line feed, hex 0A), two hex digits, or a literal character. The usual character escape sequences (e.g. \ **r** for carriage return) may be used, although they must be quoted to avoid interpretation by the shell. Specifying **—eom** is equivalent to specifying **eom EOI**.

htime *secs* Specifies the time allowed for each byte of data before returning a timeout error. Specifying **—htime** or **htime 0** disables the timer.

ptime *secs* Specifies the time allowed for an instrument to respond during a serial poll. Specifying **—ptime** or **ptime 0** disables the timer.

[—]**dma** Enables [disables] DMA hardware, if present. (This flag is provided primarily for diagnostic use.)

[—]**poll** Enables [disables] automatic polling of this instrument. See *gins*(4) for information about the auto–polling feature.

[—]**sc** System controller. This interface is [is not] the system controller.

[—]**std1** Short T1 delay. The interface T1 delay is [is not] shortened from 2.2 us to 1.2 us. See IEEE 488–1980.

[—]**tcs** Take control synchronously. The interface will [will not] synchronize assertion of ATN with the GPIB handshake to avoid loss of data. See IEEE 488–1980.

[—]**vstd1** Very short T1 delay. The interface T1 delay is [is not] shortened to 600 ns on the second and following bytes of any device–dependent message. See IEEE 488–1980.

[—]**weoi** By default, this is turned on. If it is turned off (**—weoi**), the driver will not assert EOI with the last byte of the message.

EXAMPLES

Configure the built–in interface as a system controller at GPIB address 30.

```
$ gpconf gpib0 addr 30 sc
gpib0 slot 0 addr 30 eom EOI sc tcs
```

Add an instrument "fg5010" at address 24 and an instrument "dm5010" at address 16 to the interface in option slot 2. If the interface isn't already configured, **gpconf** sets it to the default configuration.

```
$ gpconf fg5010 slot 2 addr 24 poll
gpib2 slot 2 addr 0 eom EOI sc tcs
fg5010 slot 2 addr 24 eom EOI poll
$ gpconf dm5010 slot 2 addr 16 poll
dm5010 slot 2 addr 16 eom EOI poll
```

Print the system configuration.

```
$ gpconf -a
========== slot 0 ==========
gpib0 slot 0 addr 30 eom EOI sc tcs
```

```
========== slot 2 ==========
gpib2 slot 2 addr 0 eom EOI sc tcs
dm5010 slot 2 addr 16 eom EOI poll
fg5010 slot 2 addr 24 eom EOI poll
```

FILES

/etc/gpibconf	current configuration
/etc/gpibnew	temporary file
/dev/gpib n	GPIB interface device
/dev/gpid n	GPIB configuration device
/dev/ instrument	GPIB instruments

DIAGNOSTICS

device slot *n* addr *n* ...
: The named device has been added to the configuration.

device slot *argument* : invalid slot number
"*device* addr *argument* : invalid address"
"*device* eom *argument* : invalid argument"
"*device* htime *argument* : invalid time"
"*device* ptime *argument* : invalid time"
"*device option*?"
: **Gpconf** couldn't understand your command. Correct the syntax and try again.

you must specify a GPIB address for *device*
: You must specify the GPIB address when you add a new instrument. (Only the interface address defaults to 0.)

slot *n* address *n* already in use
: There is already some device configured at this address. Correct the configuration or use a different address.

device : file or device already exists
: There is already a (non-GPIB) device or file by that name. Use a different name.

device slot *n* no interface
: There is no GPIB interface card in the specified slot. The device will be entered in the configuration file but won't be usable until you plug in an interface or specify a different slot number.

device slot *n* not configured
: The named interface is present but hasn't been configured.

can't create *device* : too many logical units in slot *n*
: You have tried to attach more than 15 instruments to one interface. You may address additional instruments through the *interface* device **gpib***n*, but cannot configure them as independent devices.

Other messages are system errors. See *intro*(2) for more information on these.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

When configuring an instrument, **gpconf** attempts to ensure that the specified interface is configured first. This will cause address conflicts if you attempt to configure an instrument at address 0 before setting the interface address.

There is no way to change the name of a device short of removing and reconfiguring it.

SEE ALSO

gpinit(1), gprm(1), gpstat(1), gpib(4), gpid(4), gins(4), config(8).

NAME

gpinit — initialize GPIB

SYNOPSIS

gpinit —a
gpinit *device* \h'0.4n'. . .

DESCRIPTION

Gpinit sets the named *devices* to their stored configuration (set by *gpconf*(1)). **Gpinit —a** is invoked during system startup to reset all GPIB devices to their previous configuration.

EXAMPLES

Initialize the interface in slot 2 and attached instruments.

```
$ gpinit gpib2
gpib2 slot 2 addr 0 eom EOI sc tcs
dm5010 slot 2 addr 16 eom EOI poll
fg5010 slot 2 addr 24 eom EOI poll
```

FILES

/etc/gpibconf	current configuration
/dev/gpibn	GPIB interface device
/dev/gpidn	GPIB configuration device
/dev/instrument	GPIB instruments

DIAGNOSTICS

device slot *n* addr *n* . . .
The named device has been initialized.

device : file or device already exists
There is already a (non–GPIB) device or file by that name. Use *gprm*(1) and *gpconf*(1) to change the name.

device slot *n* no interface
There is no GPIB interface card in the specified slot.

Other messages are system errors. See *intro*(2) for more information on these.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

gpconf(1), gprm(1), gpstat(1), gpib(4), gpid(4), gins(4), config(8).

NAME

gprm — remove GPIB instrument

SYNOPSIS

gprm *device* \h'0.4n'...

DESCRIPTION

Gprm removes the specified GPIB device(s) from the configuration. If *device* is an *interface* (**gpib***n*), any attached instruments are removed first.

EXAMPLES

Remove the device "fg5010" from the configuration.

```
$ gprm fg5010
fg5010 removed
```

FILES

/etc/gpibconf	current configuration
/etc/gpibnew	temporary file
/dev/gpibn	GPIB interface device
/dev/gpidn	GPIB configuration device
/dev/instrument	GPIB instruments

DIAGNOSTICS

device removed The named device has been removed from the configuration.

Other messages are system errors. See *intro*(2) for more information on these.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

gpconf(1), gpinit(1), gpstat(1), gpib(4), gpid(4), gins(4), config(8).

NAME

gprof — display call graph profile data

SYNOPSIS

gprof [options] [a.out [gmon.out ...]]

DESCRIPTION

gprof produces an execution profile of C, Pascal, or FORTRAN77 programs. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file (*gmon.out* default) which is created by programs which are compiled with the **—pg** option of *cc*, *pc*, and *f77*. That option also links in versions of the library routines which are compiled for profiling. The symbol table in the named object file (*a.out* default) is read and correlated with the call graph profile file. If more than one profile file is specified, the **gprof** output shows the sum of the profile information in the given profile files.

First, a flat profile is given, similar to that provided by *prof*(1). This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time.

Next, these times are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. A second listing shows the functions sorted according to the time they represent including the time of their call graph descendents. Below each function entry is shown its (direct) call graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendents is propagated to its (direct) call graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

OPTIONS

—a suppresses the printing of statically declared functions. If this option is given, all relevant information about the static function (*e.g.*, time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the *a.out* file.

—b supresses the printing of a description of each field in the profile.

—c the static call graph of the program is discovered by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.

—e *name*
suppresses the printing of the graph profile entry for routine *name* and all its descendants (unless they have other ancestors that aren't suppressed). More than one **—e** option may be given. Only one *name* may be given with each **—e** option.

—E *name*
suppresses the printing of the graph profile entry for routine *name* (and its descendants) as **—e**, above, and also excludes the time spent in *name* (and its descendants) from the total and percentage time computations. (For example, **—E** *mcount* **—E** *mcleanup* is the default.)

—f *name*
prints the graph profile entry of only the specified routine *name* and its descendants. More than one **—f** option may be given. Only one *name* may be given with each **—f** option.

—F *name*
prints the graph profile entry of only the routine *name* and its descendants (as **—f,** above) and also uses only the times of the printed routines in total time and percentage computations. More than one **—F** option may be given. Only one *name* may be given with each **—F** option. The **—F** option overrides the **—E** option.

—s a profile file *gmon.sum* is produced which represents the sum of the profile information in all the specified profile files. This summary profile file may be given to subsequent executions of **gprof** (probably also with a **—s**) to accumulate profile data across several runs of an *a.out* file.

—z displays routines which have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the **—c** option for discovering which routines were never called.

FILES

a.out	namelist and text space
gmon.out	dynamic call graph and profile
gmon.sum	summarized dynamic call graph and profile

CAVEATS

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. We assume that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

Parents which are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

The profiled program must call *exit*(3c) or return normally for the profiling information to be saved in the gmon.out file.

SEE ALSO

monitor(3c), profil(2), cc(1), prof(1).

NAME

gpstat — GPIB status

SYNOPSIS

gpstat *device* \h'0.4n'. . .

DESCRIPTION

Gpstat reports the status of the specified GPIB device(s).

FILES

/dev/gpib n	GPIB interface device
/dev/instrument	GPIB instruments

RETURN VALUE

[NO_ERR]	
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Gpstat clears the device's interrupt status.

SEE ALSO

gpconf(1), gpinit(1), gprm(1), gpib(4), gpid(4), gins(4), config(8).

NAME

graphics — access the UTek Graphics Tools

SYNOPSIS

graphics [**—r**]

DESCRIPTION

Graphics prefixes the path name **/bin/graf** to the current **$PATH** value, changes the primary shell prompt to ^, and executes a new shell (see *sh(1sh)*). The directory **/bin/graf** contains all of the UTek Graphics Tools commands. To see a list of the Graphics Tools commands, type **whatis** after entering the **graphics** shell.

If the **—r** option is given, access to the graphical commands is created in a restricted environment; that is, **$PATH** is set to

```
:/bin/graf:/rbin:/usr/rbin:/bin:/usr/bin
```

and the limited shell, *lsh(1)*, is invoked.

To restore the environment that existed prior to issuing the **graphics** command, type your *eof* character (usually ⟨CTRL–D⟩). To log off from the graphics environment, type **quit**. (**quit** only works if your login shell is *sh(1sh)*.)

OPTIONS

—r Invokes a restricted shell.

SEE ALSO

UTek Graphics Tools in *UTek Tools, Volume 2*

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), lsh(1), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), sh(1sh), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g)* in the *UTek Command Reference*.

NAME

grep — search a file for a pattern

SYNOPSIS

grep [**—E**] [**—c**] [**—h**] [**—i**] [**—l**] [**—n**] [**—s**] [**—v**] [**—w**] [**—y**] [**—e**] *pattern* [*filename ...*]

DESCRIPTION

Grep searches 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 *ex*(1); it uses a compact nondeterministic algorithm.

The filename is shown if there is more than one input file (see the **—h** option). Care should be taken when using the characters $ * [^ ¦ ( ) < > and \ in the *expression* as they are also meaningful to the Shell. It is safest to enclose the entire *expression* argument in single quotes.

OPTIONS

—E Print matching lines in the form :
filename, **line** *linenumber* : matching line

—c Only a count of matching lines is printed.

—e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a —.

—h Suppresses printing of file names when more than one file name is given.

—i, —y
Ignore case of characters specified as lower case. This option transforms lower case letters that are not inside of brackets into an expression of the form '[Xx]'. For example, the expression 'AbcD [xyz]' is transformed into 'A[Bb][Cc]D [xyz]'.

—l The names of files with matching lines are listed (once) separated by newlines. If this option is given when reading from standard input, **grep** simply exits with a value of 1.

—n Each line is preceded by its relative line number in the file.

—s Silent mode. No error message is printed for nonexistent files.

—v All lines but those matching are printed.

—w
The entire expression is treated as though it were enclosed in \<. . . \>. This syntax, which is also available to **grep** expressions, causes the expression to be treated as a 'word' (see *ex(1)*)

EXAMPLES

The following example prints all lines in the file *example* which contain the word "This" at the beginning of the line and a '.' at the end of the line. Note that the word "This" must be followed by spaces and/or tabs (^I) in this case, so a pattern like "Thise" will not be matched.

```
grep ´^This[ ^I]*.*.´ example
```

This example prints the names of all of the files in the current directory which contain the characters 'a', 'b', or 'x'. Each line found is preceded by the name of the file it was found in.

```
grep -l ´[abx]´ *
```

This example searches for the word 'john' in the file */etc/passwd*. In the first invocation, all lines containing 'john' are printed, such as those containing 'johnny'. In the second and third invocations, only those lines containing 'john' with no alphabetic characters surrounding it are found.

```
grep ´john´ /etc/passwd

grep -w ´john´ /etc/passwd

grep ´\<john\>´ /etc/passwd
```

RETURN VALUE

[0]	No errors occurred and at least one match was found.
[USAGE]	Incorrect command line syntax. Execution terminated.
[1]	No errors occured but no matches were found.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

If one of the input files is the same as the output, as in the example "egrep re * > out", that input file is not searched in order to prevent problems. No message is printed in this case. If the old functionality is required, pipe the output through *cat(1)*.

Lines are limited to 1024 characters; longer lines are truncated.

The precedence of the output specification options is **—c, —E,** and **—l,** which turn off the **—n** option and the printing of the file name and matching line.

Tests show that *egrep(1)* is the fastest of the pattern searching commands.

SEE ALSO

egrep(1), error(1), ex(1), fgrep(1), regcmp(1), sed(1), sh(1sh).

NAME

groups — show group memberships

SYNOPSIS

groups [**user**]

DESCRIPTION

The **groups** command shows the groups to which you or the optionally specified user belong. Each user belongs to a group specified in the password file */etc/passwd* and possibly to other groups as specified in the file */etc/group.*

EXAMPLES

The following invocation shows the groups to which you belong.

```
groups
```

FILES

/etc/passwd, /etc/group.

RETURN VALUE

[NO_ERRS] Command completed without error.

CAVEATS

If you do not own a file but belong to the group by which it is owned then you are granted group access to the file.

When a new file is created it is given the group of the containing directory.

SEE ALSO

chgrp(1), setgroups(2), getgrent(3c).

NAME

gtop — GPS to **plot** filter

SYNOPSIS

gtop [—**r***n*] [—**u**] [*GPS file ...*]

DESCRIPTION

Gtop transforms a GPS into commands displayable by **plot** and **tplot** filters. Input is taken from *file* if given, else from the standard input. GPS objects are translated if they fall within the window that circumscribes the first *file* unless the **–r** or **–u** option is given. Output is to the standard output.

Plot is a graphics utility and library that is part of 4.2 BSD UNIX. In System V UNIX, it is known as **tplot**. Neither is included with the current release of UTek. **Gtop** is included in the UTek Graphics Tools for compatibility with these other versions of UNIX.

OPTIONS

—**r***n*
Translate objects in GPS region *n*.

—**u** Translate all objects in the GPS universe.

EXAMPLES

The following example translates the GPS in file A.g into **plot** input.

```
gtop A.g
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

hardcopy — send "make copy" characters to Tektronix 4010 Series terminal

SYNOPSIS

hardcopy

DESCRIPTION

When issued from a Tektronix 4010 Series display terminal with a hard copy unit, **hardcopy** generates a screen copy on the unit.

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

hash — execution path hashing (*sh* built-in)

SYNOPSIS

hash [**—r**] [*name* . . .]

DESCRIPTION

For each *name,* the location in the execution search path of the command specified by *name* is determined and remembered by the shell. This speeds up execution of commands greatly.

The **—r** option causes the shell to forget about all known locations.

If no arguments are given, information about each command that is remembered is printed. The column labelled *hits* is the number of times that each listed command has been invoked by the current shell. The column labelled *cost* is a measure of the work required to locate the command in the search path.

In certain situations, the stored location of a command must be recalculated. Commands for which this will be done are indicated by an asterisk adjacent to the *hits* information. The *cost* information is incremented when the recalculation is done.

OPTIONS

—r Removes all remembered commands from the hash list.

EXAMPLES

Executing the command

```
hash ls
```

will cause the path of the command *ls* to be stored for faster execution.

RETURN VALUE

[NO_ERRS] Command completed without error.

[NO_CMD] Shell could not find the command.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hashstat(1csh), login(1), pwd(1sh), read(1sh), readonly(1sh), rehash(1csh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unhash(1csh), unset(1sh), wait(1sh), which(1sh), execve(2).

NAME

head — give first few lines

SYNOPSIS

head [—*count*] [—**f**] [*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.

If more than one file is specified, the header

= = >*filename* <= =

is printed before each file is displayed.

OPTIONS

—*count*
count linest are displayed.*count* is a decimal integer.

—**f** When reading from standard input, all input is read. The prevents the broken pipe message from the shell.

EXAMPLES

The following example displays the first 20 lines of each of the files *examp.c*, *example.h*, and */etc/passwd*.

```
head -20 examp.c example.h /etc/passwd
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Because **head** buffers its input, the command

```
(head -1; head -1) < /etc/passwd
```

does not list the first two lines of /etc/passwd. The first **head** command reads a 8192 byte buffer and prints the first line. The second **head** command must therefore start at byte 8193 of the file.

SEE ALSO

line(1), tail(1).

NAME

help — interactive manual page browser and general help facility

SYNOPSIS

help [**—p**] [[*section*] *title*]

DESCRIPTION

With no *title* argument, **help** executes a special set of menus which give general information about using commands in the system.

When a *title* (and possibly a *section* argument) is given, the command *man(1man)* is executed in order to get the name of the file containing the formatted manual entry. Because of this, the *section* argument can be in any of the valid forms for the **man** command. This manual entry and it index format table (see *manindex(5man)*) are used to display sections of the manual page. The program has special facilities for maintaining a stack during the session and for "taking notes" on a manual entry.

Screen Formats

Help is a screen–oriented manual page browser with two screen formats. The main screen is divided into three sections: Commands/Information, NAME section, and Current Section.

The Commands/Information section has three lines. The left side of first line lists the current manual entry being viewed. The right side of this line lists the top element on the stack followed by the number of items on the stack in brackets ([*n*]). The left side of the second line is where commands are entered. The right side of this line lists the available section commands for the current manual entry, with the command for the section being viewed in standout mode. The third line of the Commands/Information section is used to print error and information messages. These messages are printed in standout mode.

The NAME section of the main screen displays the NAME section from the manual entry. This section lists the names of the items described by the entry and a short description of the items.

The Current Section portion of the main screen lists the name of the section, the number of the screen page being viewed, and the total number of screen pages in the section.

The other screen format is called the "alternate screen". This screen has two sections: Help and Information.

The alternate screen Help section lists the commands available in that screen on the first two lines, and error messages on the third line.

The alternate screen Information section is used to describe the commands in the help system (via the '?' command) and to list the contents of the stack (using the 'ps' command).

Commands

There are five types of commands in the help system: Section commands, Movement commands, Notes commands, Stack commands, and Other commands. Commands may be one or two characters long and are not

followed by a return (the system knows when a valid command has been entered). Any character entered that could not be part of a command will cause the terminal bell to be rung. Once the first command character of a two-character command is entered, a backspace will erase that character.

Section Commands
The section commands cause the different sections of the manual to be displayed. The only valid section commands for a given manual entry are those listed on the right side of the second line of the Command/Information screen section. The following are the available section commands:

sy View SYNOPSIS section.

de View DESCRIPTION section.

op View OPTIONS section.

ex View EXAMPLES section.

fi View FILES section.

va View VARIABLES section.

rv View RETURN VALUE section.

di View DIAGNOSTICS section.

ca View CAVEATS section.

se View SEE ALSO section.

re View REFERENCES section.

no View notes on current entry (see Notes Commands below).

None of these commands are available in the alternate screen.

Movement Commands
The movement commands are used to move around in a manual entry section or go to the next or previous section in the entry (the order of the sections is the order listed above in the Section Commands list). Any attempt to move past the start or end of a section causes an error message, except where noted. Movement is limited to pages and half-pages. A page is the number of lines available on the screen to print the current section, and depends on the number of lines on the screen and the number of lines in the NAME section. The following are the available movement commands:

⟨**CR**⟩
Go to next section in order given above.

— Go to previous section in order given above.

⟨**sp**⟩
Go to next page in current section. If the current page is the last page, go on to the next section in the order given above.

^F Go forward one page.

^B Go back one page.

^D Go forward one half–page.

^U Go back one half–page.

1G Go to first page in section.

G Go to last page in section.

^L Redraw screen.

The following commands are also available in the alternate screen: ⟨sp⟩, ^F, ^B, ^D, ^U, and ^L.

Notes Commands The help system gives the user the ability to store and view his/her own notes on a given manual entry, such as special examples or known problems. The notes are stored in the directory *$HOME/.helpnotes* (which is created if need be) under the same name as the manual entry they correspond to. For example, the notes file for the manual entry *test(1sh)* would be stored in the file *$HOME/.helpnotes/test.1sh*. If there are notes on a manual entry, they can be displayed by using the Section command 'no'. Notes can be appended to or edited using the following commands:

an Append to notes on current entry.

en Edit notes on current entry.

The editor to use is determined in the following fashion: If the environment variable HELPEDIT is set, that command is used. If not, the variables EDITOR and EDIT are used, in that order. If none of these are set, *vi(1)* is used.

When either of these commands is invoked, the terminal mode is restored to its original value and the cursor is placed at the bottom of the screen.

These commands are not available in the alternate screen.

Stack Commands
The help system maintains a stack of up to 300 items. Each item in the stack lists the manual entry name, the current section, and the current page in the section. This allows the user to save contexts and return to that context easily. The top element on the stack is listed on the right side of the top line in the Commands/Information section of the main screen. The available stack commands are:

pu Push current page/section on stack.

po Pop stack. Top element becomes current page/section.

sw Swap current page/section for top of stack. This is especially useful for switching back and forth between two manual entries.

cs Clear stack.

ps Print stack values. The stack items are printed in order from the top

of the stack on the alternate screen. The entry name, section, and page are all printed.

These commands are not available in the alternate screen.

Other Commands

There are three commands that do not fall into the above categories. The command '?' prints a description of all of the available commands and error messages in the alternate screen.

From the main screen, the command 'q⟨CR⟩' ('q' followed by a carriage return) causes the help system to exit. From the alternate screen, the 'q' command returns the user to the main screen. Interrupting the help system at any point will terminate the session.

The command 'he' is used to look at another manual entry. The command prompts the user for the new entry specification by printing the prompt "Entry: " on the second line of the Commands/Information section of the main screen. The entry specification must be less than 67 characters long, be terminated by a return, and be in one of the following formats:

⟨*title*⟩	(eg, "test")
⟨*section*⟩ ⟨*title*⟩	(eg, "1sh test")
⟨*title*⟩(⟨*section*⟩)	(eg, "test(1sh)")

Since many manual entries are simply links to other manual entries (this is done when a manual entry describes more than one item), when the 'he' command finds that the new manual entry is the same as the current entry, no change takes place.

OPTIONS

—p All 'he' commands are automatically preceded by an invocation of the 'pu' command, thus saving the last section viewed in each manual entry. This is useful for switching back and forth between the current and previous manual entry.

EXAMPLES

The following example will execute the help system with the manual entry for the command *co(1rcs)* with all 'he' commands interpreted as 'pu' and 'he'.

```
help -p 1rcs co
```

In addition, if there are no other manual pages for **co** in section 1, the following command will work the same way.

```
help -p 1+ co
```

FILES

$HOME/.helpnotes	Manual entry notes directory.

VARIABLES

HOME	The user's home directory.
HELPEDIT	The editor to be used to edit notes.
EDITOR	The editor to be used if HELPEDIT is not set.
EDIT	The editor to be used if HELPEDIT and EDITOR are not set.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.

CAVEATS

There is no way to ask for more than one manual entry on the command line.

The terminal must have at least 80 columns and 12 lines to use the **help** command.

Since the system uses *curses(3t)*, there is only one form of standout mode. Alternatives are being studied to take advantage of underline and bold terminal modes.

Since the system uses *curses(3t)*, Tektronix 4025 terminals only work correctly when the termcap entry 4025-cr is used.

SEE ALSO

apropos(1man), buildif(1man), echo(1sh), makewhatis(1man), man(1man), manintro(1man), section(1man), whatis(1man), man(5man), manindex(5man), whatis(5man), catman(8man).

NAME

hilo — find high and low values

SYNOPSIS

hilo [**—h**] [**—l**] [**—o**] [**—ox**] [**—oy**] [*vector ...*]

DESCRIPTION

Output is the high and low values across all of the input *vector(s)*. If no *vector* is given, the standard input is assumed.

Options specify the output format. If no options are used, the output is

```
low=n     high=n
```

If the **–h** or **–l** option is used without any of the **–o** options, the output is a single number: the high or low value. If the **–h** and **–l** options are both used without any of the **–o** options, the output is a vector with two elements: the lowest value followed by the highest value.

OPTIONS

—h Only output high value.

—l Only output low value.

—o Output high, low values in "option" form (see *plot(1g)*).

—ox
Output high, low values in option form with the letter x prepended.

—oy
Output high, low values in option form with the letter y prepended.

EXAMPLES

The following example outputs the lowest value in *vectors* A and B with **xl** prepended, i.e. **xl***owvalue*.

```
hilo -ox,l A B
```

The following example uses the highest and lowest values of the *vector* A as upper and lower Y tick limits to **plot**. For example, if the highest value in A is 9 and the lowest value in A is 2, this command is equivalent to **plot –yl2,yh9 A**.

```
plot -`hilo -oy A` A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g)*.

NAME

hist — build a histogram

SYNOPSIS

hist [**—a**] [**—b**] [**—f**] [**—g**] [**—r***n*] [**—x***n*] [**—y***n*] [**—xa**] [**—ya**] [**—yl***n*] [**—yh***n*] [*vector* ...]

DESCRIPTION

Output is a GPS that describes a histogram display. Input is a *vector* of odd rank, with odd elements being bucket limits and even elements being bucket counts (see *bucket(1g)*). If no *vector* is given, the standard input is assumed.

OPTIONS

—a Suppress axes.

—b Plot histogram with bold weight lines, otherwise use medium.

—f Do not build a frame around plot area.

—g Suppress background grid.

—r*n*
Put the histogram in GPS region *n*, where *n* is between 1 and 25 inclusive.

—x*n*
Position the histogram in the GPS universe with x-origin at *n*.

—y*n*
Position the histogram in the GPS universe with y-origin at *n*.

—xa
Do not label x-axis.

—ya
Do not label y-axis.

—yl*n*
n is the y-axis low tick value.

—yh*n*
n is the y-axis high tick value.

EXAMPLES

The following example outputs the histogram described by *vector* A and locates it in region 5 of the GPS universe, with no y-axis labels.

```
hist -r5,ya A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

history — history substitution (**csh** built–in)

SYNOPSIS

history [**—h**] [**—r**] [*number*]

DESCRIPTION

The **history** command is used to print out previously executed commands in *csh(1csh)*. With no arguments, the commands are printed preceded by history numbers. If a *number* argument is given, that many items are printed. By default, the value of the shell variable 'history' is used. This document describes the **csh** history mechanism.

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence. History substitutions begin with the character ! and may begin *anywhere* in the input stream (with the provision that they *do not* nest.) This ! may be preceded by a backslash (\) to prevent its special meaning; for convenience, a ! is passed unchanged when it is followed by a blank, tab, newline, =, or (. (History substitutions also occur when an input line begins with ↑. This special abbreviation will be described later.) Any input line which contains history substitution is echoed on the terminal before it is executed, as it could have been typed without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. Their size is controlled by the **history** variable; the previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

For example, consider the following output from the **history** command:

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

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing an ! (or the current main history character) in the prompt string.

With the current event 13 you can refer to previous events by event number **11**, relatively as in **—2** (referring to the same event), by a prefix of a command word as in **!d** for event 12 or **!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*.

To select words from an event you can follow the event specification by a colon (:) and a designator for the desired words. The words of a input line are numbered from 0, the first word (usually command) being 0, the second word (first argument) being 1, and so forth. The basic word designators are:

0	first (command) word
n	*n*'th argument
↑	first argument; for example, 1
$	last argument
%	word matched by (immediately preceding) ?*s*? search
x—*y*	range of words
—*y*	abbreviates 0—*y*
*	abbreviates ↑—$, or nothing if only 1 word in event
*x**	abbreviates *x*—$
x—	like *x** but omitting word $

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

h	Remove a trailing pathname component, leaving the he
r	Remove a trailing *.xxx* component, leaving the root nar
e	Remove all but the extension *.xxx* part
s*l*/*r*/	Substitute *l* for *r*
t	Remove all leading pathname components, leaving the
&	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 substitu
x	Like q, but break into words at blanks, tabs and newlin

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

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

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a ↑. This is equivalent to **!:s**↑ and provides 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 can 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**.

The shell variable 'savehist' may be set to a number which specifies how many history items are to be saved when the user logs out. The next login will cause this list to be loaded into the history list. See the manual page for *set(1csh)* for more information.

OPTIONS

—h Display the history list without numbers. This can be used to produce files to use with the command **source —h .**

—r Print the list in reverse order. Normally, the most recent commands are printed last.

EXAMPLES

The following example prints up to the last 20 elements of the history list without numbers.

```
history -h 20
```

CAVEATS

Before a command line is saved in the history list, it is broken up into words. This means that a command such as "echo foo;bar" is saved as "echo foo ; bar", so a subsitution such as "^o;b^o:b^" will fail.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), ed(1), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

homeof — print home directory of user

SYNOPSIS

homeof [*username* ...]

DESCRIPTION

Homeof prints the home directory of each user listed on it command line. If no user name is listed, nothing is printed.

EXAMPLES

The following example prints the home directory of joeblow.

```
homeof joeblow
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.

NAME

hostid — set or print identifier of current host system

SYNOPSIS

hostid [**Internet Address**]

DESCRIPTION

The **hostid** command prints the identifier of the current host as an internet address.

The command *netconfig(8n)* ensures that hostid is set correctly whenever the workstation is rebooted. The hostid is saved in the *network.conf(5n)* file.

Only the superuser can set the hostid. The input format is the standard internet address format, see *netconfig(8n)*. The hostid can be set to the internet address of any of the interfaces (typically there is only one interface, so there is no choice).

The value set by **hostid** is also returned as part of the stat structure (see *stat(2)*) to aid in determining the location of a file.

FILES

/etc/network.conf	File that remembers network configuration upon rebooting.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	You incorrectly specified an internet address.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

gethostid(2), sethostid(2), stat(2), netconfig(8n), network.conf(5n).

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 done automatically at startup by */etc/rc.net,* using *netconfig(8n).* The hostname is stored in */etc/network.conf* file.

If the hostname is changed, *shutdown(8)* the system and bring it back to multi–user again to restart the network daemons.

FILES

/etc/network.conf current host data

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

SEE ALSO

hostid(1n). gethostname(2), sethostname(2), network.conf(5n), netconfig(8n).

NAME

hpd — display GPS on a a Hewlett Packard 7221A Graphics Plotter

SYNOPSIS

hpd [—**c***n*] [—**p***n*] [—**r***n*] [—**s***n*] [—**u**] [—**xd***n*] [—**xv***n*] [—**yd***n*] [—**yv***n*] [*GPS file ...*]

DESCRIPTION

Output is device code for a Hewlett Packard 7221A Plotter. A viewing window is computed from the maximum and minimum points in the GPS *file*(s) unless the **r** or **u** option is used. If no *file* is given, the standard input is assumed.

OPTIONS

—**c***n*
Select character set *n*, *n* between 0 and 5 (see the HP 7221A Plotter Operating and Programming Manual, Appendix A).

—**p***n*
Select pen numbered *n*, *n* between 1 and 4 inclusive.

—**r***n*
Window on GPS region *n*, *n* between 1 and 25 inclusive.

—**s***n*
Slant characters *n* degrees counterclockwise from the vertical.

—**u** Window on the entire GPS universe.

—**xd***n*
Set x displacement of the viewport's lower left corner to *n* inches.

—**xv***n*
Set width of viewport to *n* inches.

—**yd***n*
Set y displacement of the viewport's lower left corner to *n* inches.

—**yv***n*
Set height of viewport to *n* inches.

EXAMPLES

The following example displays A.g and B.g together using pen number three and slanting characters forward by 20 degrees. The viewing window is built to include all of both files, and the device viewport is 12.5 by 10 inches.

```
hpd -p3,s-20,xv12.5,yv10 A.g B.g
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

id — print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

Id writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

SEE ALSO

who(1n), whoami(1), getuid(2).

NAME

ident — identify files

SYNOPSIS

ident [**—k** *list*] [**—n** *number*] *filename ...*

DESCRIPTION

Ident searches the named files for all occurrences of the pattern *$keyword:...$,* where *keyword* is one of

Author
Date
Header
Locker
Log
Revision
Source
State

These patterns are normally inserted automatically by the RCS command *co(1rcs),* but can also be inserted manually.

Ident works on text files as well as object files.

OPTIONS

—k *list*
Search only for the keywords in *list*. The keywords in *list* must be separated by commas.

—n *number*
Stop searching the input file when *number* keywords have been found.

EXAMPLES

If the C program in file f.c contains

```
char rcsid[] = "$Header: Header information $";
```

and f.c is compiled into f.o, then the command

```
ident f.c f.o
```

will print

```
f.c:
     $Header: Header information $
f.o:
     $Header: Header information $
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

The maximum number of revisions that can be stored in a single RCS file is 719. When there are more than 700 revisions in a file, a warning message is printed on the terminal (if possible) every time an RCS command works on the file. See the manual page for *rcsfile(5rcs)* for information on what action to take in this case.

SEE ALSO

ci(1rcs), co(1rcs), rlog(1rcs), rcs(1rcs), rcsdiff(1rcs), rcsintro(1rcs), rcsmerge(1rcs), rcsfile(5rcs).

NAME

inc — incorporate new mail

SYNOPSIS

inc [+**folder**] [—**audit** *audit-file*] [—**ms** *ms-folder*] [—**help**] [—**changecur**] [—**nochangecur**] [—**time**] [—**notime**] [—**numdate**] [—**nonumdate**]

DESCRIPTION

Inc incorporates mail from the user's incoming mail drop *(/usr/spool/mail/$USER)* into an MH folder. If '+folder' isn't specified, the folder named *inbox* in the user's MH directory will be used. The new messages being incorporated are assigned numbers starting with the next highest number in the folder. If the specified (or default) folder doesn't exist, the user will be queried prior to its creation. As the messages are processed, a *scan* listing of the new mail is produced.

If the user's profile contains a 'Msg—Protect: nnn' entry, it will be used as the protection on the newly created messages, otherwise the MH default of 664 will be used. During all operations on messages, this initially assigned protection will be preserved for each message, so *chmod*(1) may be used to set a protection on an individual message, and its protection will be preserved thereafter.

If the switch **—audit audit-file** is specified (usually as a default switch in the profile), then **inc** will append a header line and a line per message to the end of the specified audit-file with the format:

```
⟨⟨inc⟩⟩ date
        ⟨scan line for first message⟩
        ⟨scan line for second message⟩
                        ⟨etc.⟩
```

This is useful for keeping track of volume and source of incoming mail. Eventually, *repl*, *forw*, and *comp* may also produce audits to this (or another) file, perhaps with 'Message-Id:' information to keep an exact correspondence history. 'Audit-file' will be in the user's MH directory unless a full path is specified.

Inc will incorporate even illegally formatted messages into the user's MH folder, inserting a blank line prior to the offending component and printing a comment identifying the bad message.

In all cases, the */usr/spool/mail/$USER* file will be zeroed.

Your .mh_profile can contain the following entries:

Path: To determine the user's MH directory
Folder—Protect: For protection on new folders
Msg—Protect: For protection on new messages

inc has the following defaults:

+*folder* defaults to *inbox*

The folder into which the message is being incorporated will become the current folder, and the first message incorporated will be the current message. This leaves the context ready for a *show* of the first new message unless **—nochangecur is set.**

OPTIONS

Options with an astrisk following indicates default condition.

—audit *audit-file*
inc will append a header line and a line per message to the end of the specified audit–file with the format:

⟨⟨inc⟩⟩ date
⟨scan line for first message⟩
⟨scan line for second message⟩
⟨etc.⟩

—help
Displays a synopsis of the **inc** command.

+folder
folder becomes the current folder.

—ms *ms-file*
inc will read from *ms-file* as though it were the mail drop, instead of the default mail drop */usr/spool/mail/$USER*.

—changecur*
will cause the current message to be changed to the first message incorporated.

—nochangecur
will cause the current message **not** to be changed to the first message incorporated.

—time
will cause the time the message was sent (in 24 hour format) to be printed on the scan line during incorporation.

—notime*
will **not** cause the time the message was sent (in 24 hour format) to be printed on the scan line during incorporation.

—numdate
will cause the date in the scan line during incoporation to be printed in YYMMDD format (useful for sorting scan lines by dates).

—nonumdate*
will **not** cause the date in the scan line during incoporation to be printed in YYMMDD format (useful for sorting scan lines by dates).

FILES

$HOME/.mh_profile	The user profile
/usr/spool/mail/$USER	The user's mail drop
⟨*mh-dir*⟩*/audit-file*	Audit trace file (optional)

SEE ALSO

comp(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh).

NAME

intro — introduction to the UTek Graphics Tools

DESCRIPTION

The commands in section 1G comprise the UTek Graphics Tools. These include commands to generate and process numbers, commands to turn numbers into graphics, and commands to display and edit graphics on display terminals.

The Graphics Tools commands produce the best results when used with a display terminal such as a Tektronix 4010 Series or 4100 Series terminal, or a graphics terminal emulator window on a 6000 Family display. However, graphic information can be stored in a file. You can create graphics files on any terminal for later display on a graphical device.

To access the UTek Graphics Tools commands, type **graphics**. The shell variable **$PATH** is altered to include the Graphics Tools commands, the shell primary prompt is changed to ^, and a new shell is started up. This is always a Bourne shell (*sh(1sh)*), even if your login shell is the C–shell (*csh(1csh)*). To leave the **graphics** shell, type ⟨CTRL–D⟩ (or your *eof* character, if different: see *stty(1)*). To log off from within the **graphics** shell, type **quit**. (**quit** only works if your login shell is *sh(1sh)*.)

All of the Graphics Tools commands are of the same general form:

command [—*option(s)*] [*file(s)*]

If a command accepts multiple *options*, they can be specified in three ways:

—o [*value*] **—o** [*value*] ...

—o [*value*],**o** [*value*] ...

—o [*value*]**o** [*value*] ...

where **o** represents the one–letter or two–letter option name, and [*value*] represents the option's value. Some options take numeric values, some take string values, and some take no value. In a string value, any spaces or commas must be escaped (preceded by a backslash).

Generally, if the *file* argument is omitted the command reads from the standard input. A single minus sign (—) used as a filename also refers to the standard input. The type of *file* that can be used depends on the command.

Most of the numerical commands in the UTek Graphics Tools accept *vectors* as input and produce *vectors* as output. A *vector* is a text file containing numbers separated by non–numbers. Numbers are constructed in the usual way:

[*sign*]*digits*[.*digits*][**e**[*sign*]*digits*]

or

[*sign*][*digits*].*digits*[**e**[*sign*]*digits*]

where *sign* is + or −, *digits* is any series of one or more digits (0 through 9), and optional groups are surrounded by brackets. For example, the following are all legal numbers:

15 −.239 1.77e−26 +63.2e+3

Because anything that is not a number is a delimiter, the vector

```
1 melon, 2 cinnamon sticks, 3/4 cup milk, and 5 loons
```

is equivalent to the vector

```
1 2 3 4 5
```

Vectors are created by several of the UTek Graphics Tools, including *gas(1g)* (generate additive sequence) and *rand(1g)* (generate random numbers). Vectors can also be created by a text editor or any other UTek command that produces text files.

Some Graphics Tools commands, such as *plot(1g)* and *hist(1g)*, accept vectors and produce Graphical Primitive Strings, or GPS. A GPS is a text string that describes a picture. The format of a GPS is described in *gps(5g)*.

You can display a GPS on a Tektronix terminal with *td(1g)*, or on a Hewlett–Packard plotter with *hpd(1g)*. You can also edit or create a GPS with the graphical editor *ged(1g)* and get a human–readable listing of a GPS with *gd(1g)*.

All of the UTek Graphics Tools, with the exception of *graphics(1g)*, reside in */bin/graf*. If you want to use Graphics Tools commands in a shell script, the script should begin with the following lines:

```
#!/bin/sh
PATH="/bin/graf:$PATH"
export PATH
```

EXAMPLES

The following example creates a *vector* of 100 random numbers with *rand(1g)*, sorts it into ascending order with *qsort(1g)*, and counts the number of elements in a series of intervals with *bucket(1g)*. **bucket** produces a vector describing the number of elements in each interval, and *hist(1g)* turns that vector into a GPS describing a histogram. Finally, *td(1g)* displays the GPS on a Tektronix terminal.

```
rand -n100 ¦ qsort ¦ bucket ¦ hist ¦ td
```

FILES

*/bin/graf/**	Executable commands.
*/usr/lib/graf/whatis.d/**	Brief descriptions of each command for *whatis(1g)*.
*/usr/lib/graf/ttoc.d/**	*Ed(1)* scripts for *ttoc(1g)*.

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

REFERENCES

UTek Graphics Tools in *UTek Tools, Volume 2.*

NAME

jobs — c–shell job control

SYNOPSIS

jobs [**—l**]

DESCRIPTION

The **jobs** command is used to print information about jobs being run by the *csh(1csh)*. This document describes the **csh** job control facility.

Csh associates a *job* with each pipeline or command. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers (an attempt is made to keep the job numbers less than 10 by wrapping back to 1 after job 9). When a job is started asynchronously with &, the shell prints a line which looks like:

```
[1] 1234
```

indicating that the jobs which was started asynchronously was job number 1 and had one (top–level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit ⟨CNTRL Z⟩ which sends a STOP signal to the current job. The shell will then normally indicate that the job has been 'Stopped', and print another prompt. You can then manipulate the state of this job, putting it in the background with the **bg** command, or run some other commands and then eventually bring the job back into the foreground with the foreground command **fg**. A ⟨CNTRL Z⟩ takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed. There is another special key, ⟨CNTRL Y⟩, which does not generate a STOP signal until a program attempts to *read(2)* it. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command **stty tostop**. If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. The character **%** introduces a job name. If you wish to refer to job number 1, you can name it as **%1**. Just naming a job brings it to the foreground; thus **%1** is a synonym for **fg %1**, bringing job 1 back into the foreground. Similarly saying **%1 &** resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them, if these prefixes are unambiguous; thus **%ex** would normally restart a suspended *ex(1)* job, if there were only one suspended job whose name began with the string **ex**. It is also possible to say **%?***string* which specifies a job whose text contains *string,* if there is only one such job.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a + and the previous job with a — (dash). The abbreviation **%+** refers to the current job, and **%—** refers to the previous job. For close analogy with the syntax of the *history* mechanism (described in *history(1csh)*), **%%** is also a synonym for the current job.

OPTIONS

—l Print a long listing of all job information. The process id and working directory for the job are printed.

EXAMPLES

Assume that you ran the command "ex file" in the directory /usr/fred, suspended the job, and changed your directory to /. The command

```
jobs -l
```

would print a listing like:

```
[1] + 1234  Stopped    ex file (wd: /usr/fred)
```

CAVEATS

The syntax **%***job* is not expanded to the process id of the job. For this reason, commands like *renice(1)* will not work with the '%' specifiers.

Piping the output from **jobs,** as in the command line "jobs –l ¦ more", will result in no output. The reason for this is that a pipeline causes all commands, including built–ins, to be run in a subshell. Since job information is stored on a per–shell basis, a subshell will not have the same job information as the parent. It is possible to get this information by using redirection, as in "jobs –l › jobs.$$; more jobs.$$", since redirection does not cause a subshell to be used to execute **jobs.**

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), ex(1), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), renice(1), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), read(2).

NAME

join — relational database operator

SYNOPSIS

join [**—a***n*] [**—e** *s*] [**—j***n m*] [**—o** *list*] [**—t***c*] *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*, but the output format may be specified in any order by using the **—o** option.

Fields are normally separated by blank, tab or newline. In this case, multiple separators count as one, and leading separators are discarded.

OPTIONS

—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*. This must be used with the **—o** option so that the correct number of fields is known. Otherwise, the **—e** option is ignored.

—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. The elements of the list must be separate command line arguments.

—t*c*
Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The default separator whitespace (tabs and spaces). Only one separator character can be specified.

EXAMPLES

In the following example, the contents of the file *numbers* is

```
Bill 482-3659
Jane 441-3666
Sue  819-2134
```

and the contents of the file *work_area* is

```
Materials Bill
Documents Jane
Management Sue
```

The command

```
join -j1 2 work_area numbers
```

produces the output

```
Bill Materials 482-3659
Jane Documents 441-3666
Sue Management 819-2134
```

and the command

```
join -j2 2 numbers work_area
```

produces the output

```
Bill 482-3659 Materials
Jane 441-3666 Documents
Sue 819-2134 Management
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

With default field separation, the collating sequence is that of *sort —b*; with **—t**, the sequence is that of a plain sort.

An input file to **join** may have initial blank lines, but shouldn't contain any other blank lines.

SEE ALSO

awk(1), comm(1), cut(1), egrep(1), fgrep(1), grep(1), look(1), paste(1), sort(1), uniq(1).

NAME

kill — terminate a process with extreme prejudice

SYNOPSIS

kill [—*sig*] processid ...
kill —l

DESCRIPTION

Kill sends the TERM (terminate, 15) signal to the specified processes. If a signal name or number preceded by '—' is given as first argument, that signal is sent instead of terminate (see *signal*(3c)). The signal names are listed by 'kill —l', and are as given in */usr/include/signal.h,* stripped of the common SIG prefix.

The terminate signal will kill processes that do not catch the signal; 'kill —9 ...' is a sure kill, as the KILL (9) signal cannot be caught. By convention, if process number 0 is specified, all members in the process group (i.e. processes resulting from the current login) are signaled. The killed processes must belong to the current user unless he is the super-user.

To shut the system down and bring it up single user the super-user may send the initialization process a TERM (terminate) signal by 'kill 1'; see *init*(8). To force *init* to close and open terminals according to what is currently in */etc/ttys* use 'kill —HUP 1' (sending a hangup (which is signal number 1) to process number 1)

The process number of an asynchronous process started with '&' is reported by the shell.

OPTIONS

—l Lists the signal names.

—*sig*
The specified signal is sent to the specified process.

EXAMPLES

The following example absolutely terminates process number 186.

```
kill -9 186
```

This example shows a shell script that can be used to log off of the system by sending the hangup signal to the process group.

```
#!/bin/sh
echo "Logging off..."
kill -HUP 0
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

ps(1), sh(1sh), kill(2), signal(3c).

NAME

kill — send signal to jobs (**csh** built-in)

SYNOPSIS

kill [—*signal*] *process...*
kill —l

DESCRIPTION

The **kill** command sends the specified *signal* (or TERM, if none is given) to the specified processes. A *process* may either be a process id number or a specifier of the form **%***job* as described in *jobs(1csh)*. The *signal* argument may be either a signal number or a name as listed by the command **kill —l.**

There is no default process, so executing **kill** with no process arguments does not send the signal to the current job.

If the signal being sent is TERM (terminate) or HUP (hangup), the process is sent the CONT (continue) signal as well, to allow the process to handle the signal specially if it was stopped previously.

OPTIONS

—l Print a list of valid signal names. The list is printed in signal number order.

EXAMPLES

Assume that job number 1 is the command "make mycommand", and has a process id of 6477. Also, assume that there are no other *make(1)* jobs executing. The following shows nine different ways to send the terminate (which is signal number 15) signal to the process.

```
kill %1
kill %make
kill 6477
kill -TERM %1
kill -TERM %make
kill -TERM 6477
kill -15 %1
kill -15 %make
kill -15 6477
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	The process does not exist or is not owned by the user.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), kill(2), killpg(2), signal(3c).

NAME

label — label the axis of a data plot

SYNOPSIS

label [**—b**] [**—c**] [**—F***file*] [**—h**] [**—p**] [**—r***n*] [**—x**] [**—xu**] [**—y**] [**—yr**] [*GPS file ...*]

DESCRIPTION

Label appends axis labels from a *label file* to a GPS of a data plot (like that produced by *hist(1g)*, *bar(1g)* and *plot(1g)*). Each line of the *label file* is taken as one label. For **plot** labels, be sure to include **xi1** on the **plot** command line (see *plot(1g)*). Blank lines yield null labels. Either the GPS or the *label file*, but not both, may come from the standard input.

OPTIONS

—b Assume the input is a bar chart.

—c Retain lower case letters in labels, otherwise all letters are upper case.

—F*file*
file is the *label file*.

—h Assume the input is a histogram.

—p Assume the input is an x–y plot. This is the default.

—r*n*
Labels are rotated *n* degrees. The pivot point is the first character.

—x Label the x–axis. This is the default.

—xu
Label the upper x–axis, i.e., the top of the plot.

—y Label the y–axis.

—yr
Label the right y–axis, i.e., the right side of the plot.

EXAMPLES

The following example labels the file A.g, which is assumed to be an x–y plot. The labels come from the file *labs*.

```
label -Flabs A.g
```

The following example labels the file A.g on the right y–axis with labels from the standard input. The labels are printed at 45 degrees below the horizontal.

```
label -yr,r-45 A.g
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

last — indicate last logins of users and teletypes

SYNOPSIS

last [*username*...] [*tty*...]

DESCRIPTION

Last will look back in the *wtmp* file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example 'last 0' is the same as 'last tty0'. If multiple arguments are given, the information which applies to any of the arguments is printed. For example 'last root console' would list all of "root's" sessions as well as all sessions on the console terminal. **Last** will print the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, **last** so indicates.

Last with no arguments prints a record of all logins and logouts, in reverse order.

If **last** is interrupted, it indicates how far the search has progressed in *wtmp*. If interrupted with a quit signal (generated by a control-\) **last** indicates how far the search has progressed so far, and the search continues.

EXAMPLES

The following invocation will list the information described above for the tty p6. The information provided would only be for users that have logged onto the tty p6.

```
last p6
```

Since the pseudo-user **reboot** logs in at reboots of the system, the command

```
last reboot
```

will give an indication of mean time between reboot.

FILES

/usr/adm/wtmp	login data base
/usr/adm/shutdownlog	which records shutdowns and reasons for same

RETURN VALUE

[NO_ERRS]	Command completed without error.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

who(1n), finger(1n), wtmp(5).

NAME

ld — link editor

SYNOPSIS

ld [**—d**] [**—e**] [**—n**] [**—o**] [**—r**] [**—s**] [**—t**] [**—u**] [**—x**] [**—y** *sym*] [**—z**] [**—A**] [**—D**] [**—G**] [**—M**] [**—N**] [**—S**] [**—T**] [**—X**] *filename...* [**—l***x*]

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 (unless the **—e** option is specified).

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, and the library has not been processed by *ranlib*(1), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. The first member of a library should be a file named '____.SYMDEF', which is understood to be a dictionary for the library as produced by *ranlib*(1); the dictionary is searched iteratively to satisfy as many references as possible.

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.

OPTIONS

Ld understands several options. Except for **—l**, they should appear before the file names.

—d Force definition of common storage even if the **—r** flag is present.

—e The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default.

—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.

—n Arrange (by giving the output file a 0410 "magic number") 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 1024 byte boundary following the end of the text.

—o The *name* argument after **—o** is used as the name of the **ld** output file, instead of **a.out**.

—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.

—s 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debuggers). This information can also be removed by *strip*(1).

—t execution trace. Probably only useful for debugging the loader.

—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.

—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.

—y*sym*
Indicate each file in which *sym* appears, its type and whether the file defines or references it. Many such options may be given to trace many symbols. (It is usually necessary to begin *sym* with an '_', as external C, FORTRAN and Pascal variables begin with underscores.)

—z Arrange for the process to be loaded on demand from the resulting executable file (413 format) rather than preloading by default. Results in a 1024 byte header on the output file followed by a text and data segment each of which have size a multiple of 1024 bytes (being padded out with nulls in the file if necessary). With this format the first few BSS segment symbols may actually appear (from the output of *size*(1)) to live in the data segment; this to avoid wasting the space resulting from data segment size roundup.

—A
This option specifies incremental loading, i.e. linking is to be done in a manner so that the resulting object may be read into an already executing program. The next argument is the name of a file whose symbol table will be taken as a basis on which to define additional symbols. Only newly linked material will be entered into the text and data portions of **a.out,** but the new symbol table will reflect every symbol defined before and after the incremental load. This argument must appear before any other object file in the argument list. The **—T** option may be used as well, and will be taken to mean that the

newly linked segment will commence at the corresponding address (which must be a multiple of 1024). The default value is the old value of _end.

—D
Take the next argument as a hexadecimal number and pad the data segment with zero bytes to the indicated length.

—G
Take the next argument as a hexadecimal number and use it as the base address of the bss segment. (Useful for ROM based code).

—M
produce a primitive load map, listing the names of the files which will be loaded.

—N
Do not make the text portion read only or sharable. (Use "magic number" 0407.)

—S
'Strip' the output by removing all symbols except locals and globals.

—T The next argument is a hexadecimal number which sets the text segment origin. The default origin is 0.

—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.

FILES

/lib/lib.a*	libraries
/usr/lib/lib.a*	more libraries
/usr/local/lib/lib.a*	still more libraries
a.out	output file

CAVEATS

There is no way to force data to be page aligned.

SEE ALSO

as(1), ar(1), cc(1), ranlib(1).

NAME

leave — remind you when you have to leave

SYNOPSIS

leave [–i] [–pNumber] [–s] [time] [message ...]

DESCRIPTION

Leave waits until the specified *time,* then reminds you of that event. You are reminded 5 minutes and 1 minute before the actual time, at the time, and every minute for ten minutes thereafter. When you log off, **leave** exits just before it would have printed the next message.

The *time* is either in the form *+Number* where number is the delay in minutes from the current time or *hhmm,* a time in hours and minutes (on a 12 or 24 hour clock). All times given in the second form are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If no *time* is given, **leave** prompts with "When do you have to leave?". A reply of newline causes **leave** to exit, otherwise the reply is assumed to be a *time.* This form is suitable for inclusion in a *.login* or *.profile.*

A *message* string can be added to the notification to provide information on which activity you need to leave for, useful if more than one **leave** is active.

OPTIONS

—i Prints the process number for the leave invocation on standard out. Useful for getting rid of the repeating reminder. This option invokes –s automatically.

—pNumber
Changes the default pester time. Normally **leave** will remind you of your tardiness for ten minutes after the indicated time. *Number* is set to the integer pester time desired.

—s Silent mode. Suppresses the initial notification of the alarm set time. Does not stop later messages, error messages, or the prompt for *time.*

EXAMPLES

To obtain a reminder to leave for a special meeting in ten minutes without an initial message and with pester time set to 2 minutes the following is used.

```
leave -p2 -s +10 Special Meeting
```

An invocation for a lunch meeting at 12:30pm would be.

```
leave 1230 Lunch with Smith.
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.

CAVEATS

Leave ignores interrupts, quits, and terminates. To get rid of it you should either log off or use "kill —9" giving its process id.

SEE ALSO

at(1), calendar(1), signal(3c), sleep(1), ctime(3c).

NAME

lex — generator of lexical analysis programs

SYNOPSIS

lex [**—tvfn**] [*filename*] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analyis of text. The input **files** (standard input default) contain regular expressions to be searched for, and actions written in C to be executed when expressions are found.

A C source program, 'lex.yy.c' is generated, to be compiled thus:

```
cc lex.yy.c -ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized.

OPTIONS

—f "Faster" compilation: don't bother to pack the resulting tables; limited to small programs.

—n Opposite of **—v**; **—n** is default.

—t Place the result on the standard output instead of in file *lex.yy.c*.

—v Print a one–line summary of statistics of the generated analyzer.

EXAMPLES

```
lex lexcommands
```

would draw **lex** instructions from the file *lexcommands,* and place the output in *lex.yy.c*

```
%%
[A-Z] putchar(yytext[0]+'a'-'A');
[ ]+$
[ ]+ putchar(' ');
```

is an example of a **lex** program that would be put into a **lex** command file. This program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

SEE ALSO

yacc(1), sed(1).

NAME

ls, lf, lg, ll, lr, lx — list contents of directory

SYNOPSIS

ls [**—1**] [**—A**] [**—C**] [**—F**] [**—L**] [**—R**] [**—a**] [**—c**] [**—d**] [**—f**] [**—g**] [**—h**] [**—i**] [**—l**] [**—q**] [**—r**] [**—s**] [**—t**] [**—u**] [**—x**] *filename...*

DESCRIPTION

For each argument that names a directory or symbolic link to a directory (unless **—l** is specified), **ls** lists the contents of the directory; for each file argument, **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.

The commands **lf, lg, ll, lr,** and **lx** are links to **ls** which set flags as follows:

lf –F

lg –l, –g

ll –l

lr –R

lx –x

The mode printed under the **—l** option contains 11 characters which are interpreted as follows: the first character is

b if the entry is a block–type special file;
c if the entry is a character–type special file;
d if the entry is a directory;
l if the entry is a symbolic link, or
— 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. 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 the set–group–id bit set; likewise the user–execute permission character is given as **s** if the file has the set–user–id bit set.

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 is printed. If you find the size using the **—s** option, this reflects the actual amount of disk space a file consumes. In this case, a count of indirect blocks is also printed. If you find the size using the **—l** option, a count of indirect blocks is NOT included. The size obtained with **—l** is not the actual number of bytes, but an end of data location. Unless the file was created by skipping around and writing, this is the actual number of bytes consumed.

OPTIONS

—a List all entries; in the absence of this option, entries whose names begin with a period (**.**) are *not* listed.

—A
list all entries except '.' and '..'. This option is always set for the super–user.

—c Use time of the last file status change for sorting or printing. This time is the last time that the file's data or the i–node was changed (meaning the last time the data was changed or the file was linked to. See the manual page for *stat(2)* for more information).

—C
force multi–column output; this is the default when output is to a terminal.

—d If argument is a directory, list only its name; often used with **—l** to get the status of a 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.

—F cause directories to be marked with a trailing '/', symbolic links with a trailing '@', and executable files with a trailing '*'. All symbolic links to existing directories are marked with a trailing '@/'. All symbolic links to existing executable files are marked with a trailing '@*'. All symbolic links to nonexistent files are marked with a trailing '@?'.

—g Include the group ownership of the file in a long output.

—h Follow only "hard" directory paths during recursion (**—R** option). By default, symbolic links to directories are traversed (except when **—l** is used without **—L**). This option prevents these symbolic links from being traversed.

—i For each file, print the i–number in the first column of the report.

—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. If the file is a symbolic link the pathname of the linked–to file is printed preceded by ''–>''.

—L If argument is a symbolic link, list the file or directory the link references rather than the link itself.

—q force printing of non–graphic characters in file names as the character '?'; this is the default when output is to a terminal.

—r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

—R
recursively list subdirectories encountered. (Note: In order to avoid symbolic link loops, directories are only traversed the first time they are encountered, whether as a regular directory or as a symbolic link.)

—s Give size in kilobytes of each file.

—t Sort by time modified (latest first) instead of by name.

—u Use time of last access instead of last modification for sorting (with the **—t** option) and/or printing (with the **—l** option).

—x List only directories. If the **—L** option is also given, symbolic links to directories will also be listed.

—1 force one entry per line output format; this is the default when output is not to a terminal.

EXAMPLES

The following example lists all of the files in the directory */bin*.

```
ls /bin
```

FILES

/etc/passwd	to get user id's for 'ls —l'.
/etc/group	to get group id's for 'ls —g'.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The date printed in the long listing (**—l** option) will contain the time of day if the file is up to six months old or has a date up to 48 hours in the future. All other dates will print with the year instead of the time.

The option setting based on whether the output is a teletype is undesirable as *ls —s* is much different than *ls —s ¦ lpr* . On the other hand, not doing this setting would make old shell scripts which used **ls** almost certain losers.

The commands *lf(1), lg(1), ll(1), lr(1),* and *lx(1)* are not separate programs; they are links to **ls.** This means that removing these commands will not free up a lot of disk space. This also means that modifying **ls** requires re–linking the commands.

SEE ALSO

cat(1), chmod(1), find(1), rm(1).

NAME

ls, lf, lg, ll, lr, lx — list contents of directory

SYNOPSIS

ls [**—1**] [**—A**] [**—C**] [**—F**] [**—L**] [**—R**] [**—a**] [**—c**] [**—d**] [**—f**] [**—g**] [**—h**] [**—i**] [**—l**] [**—q**] [**—r**] [**—s**] [**—t**] [**—u**] [**—x**] *filename...*

DESCRIPTION

For each argument that names a directory or symbolic link to a directory (unless **—l** is specified), **ls** lists the contents of the directory; for each file argument, **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.

The commands **lf, lg, ll, lr,** and **lx** are links to **ls** which set flags as follows:

lf	–F
lg	–l, –g
ll	–l
lr	–R
lx	–x

The mode printed under the **—l** option contains 11 characters which are interpreted as follows: the first character is

b if the entry is a block–type special file;
c if the entry is a character–type special file;
d if the entry is a directory;
l if the entry is a symbolic link, or
— 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. 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 the set–group–id bit set; likewise the user–execute permission character is given as **s** if the file has the set–user–id bit set.

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 is printed. If you find the size using the **—s** option, this reflects the actual amount of disk space a file consumes. In this case, a count of indirect blocks is also printed. If you find the size using the **—l** option, a count of indirect blocks is NOT included. The size obtained with **—l** is not the actual number of bytes, but an end of data location. Unless the file was created by skipping around and writing, this is the actual number of bytes consumed.

OPTIONS

—a List all entries; in the absence of this option, entries whose names begin with a period (**.**) are *not* listed.

—A
list all entries except '.' and '..'. This option is always set for the super–user.

—c Use time of the last file status change for sorting or printing. This time is the last time that the file's data or the i–node was changed (meaning the last time the data was changed or the file was linked to. See the manual page for *stat(2)* for more information).

—C
force multi–column output; this is the default when output is to a terminal.

—d If argument is a directory, list only its name; often used with **—l** to get the status of a 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.

—F cause directories to be marked with a trailing '/', symbolic links with a trailing '@', and executable files with a trailing '*'. All symbolic links to existing directories are marked with a trailing '@/'. All symbolic links to existing executable files are marked with a trailing '@*'. All symbolic links to nonexistent files are marked with a trailing '@?'.

—g Include the group ownership of the file in a long output.

—h Follow only "hard" directory paths during recursion (**—R** option). By default, symbolic links to directories are traversed (except when **—l** is used without **—L**). This option prevents these symbolic links from being traversed.

—i For each file, print the i–number in the first column of the report.

—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. If the file is a symbolic link the pathname of the linked–to file is printed preceded by "–>".

—L If argument is a symbolic link, list the file or directory the link references rather than the link itself.

—q force printing of non–graphic characters in file names as the character '?'; this is the default when output is to a terminal.

—r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

—R
recursively list subdirectories encountered. (Note: In order to avoid symbolic link loops, directories are only traversed the first time they are encountered, whether as a regular directory or as a symbolic link.)

—s Give size in kilobytes of each file.

—t Sort by time modified (latest first) instead of by name.

—u Use time of last access instead of last modification for sorting (with the **—t** option) and/or printing (with the **—l** option).

—x List only directories. If the **—L** option is also given, symbolic links to directories will also be listed.

—1 force one entry per line output format; this is the default when output is not to a terminal.

EXAMPLES

The following example lists all of the files in the directory */bin*.

```
ls /bin
```

FILES

/etc/passwd	to get user id's for 'ls —l'.
/etc/group	to get group id's for 'ls —g'.

RETURN VALUE

[NO_ERRS] Command completed without error.

[NP_ERR] An error occurred that was not a system error. Execution terminated.

[P_WARN] A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The date printed in the long listing (**—l** option) will contain the time of day if the file is up to six months old or has a date up to 48 hours in the future. All other dates will print with the year instead of the time.

The option setting based on whether the output is a teletype is undesirable as *ls —s* is much different than *ls —s ¦ lpr* . On the other hand, not doing this setting would make old shell scripts which used **ls** almost certain losers.

The commands *lf(1), lg(1), ll(1), lr(1),* and *lx(1)* are not separate programs; they are links to **ls.** This means that removing these commands will not free up a lot of disk space. This also means that modifying **ls** requires re–linking the commands.

SEE ALSO

cat(1), chmod(1), find(1), rm(1).

NAME

limit, unlimit — set resource limitations (**csh** built-in)

SYNOPSIS

limit [*resource* [*maximum*]]
unlimit [*resource...*]

DESCRIPTION

With no arguments, **limit** prints all resource limits (see below for a list of limitable resources). With only a *resource,* the limit for that resource is printed. With a *resource* and a *maximum,* the limit for that resource is set to the given maximum, if possible. This limit will apply to the current shell and is inherited by all child processes.

The **unlimit** command removes the limit on the named resources. If no arguments are given, all resource limits are removed.

Resources that are currently controlable include *cputime* (the maximum number of CPU-seconds to be used by each process), *filesize* (the largest single file which can be created), *datasize* (the maximum growth of the data + stack region via *sbrk(2)* beyond the end of the program text), *stacksize* (the maximum size of the automatically-extended stack region), and *coredumpsize* (the size of the largest core dump that will be created).

The *maximum-use* may be given as a (floating point or integer) number followed by a scale factor. For all limits other than *cputime* the default scale is **k** or *kilobytes* (1024 bytes); a scale factor of **m** or *megabytes* may also be used. For *cputime* the default scaling is *seconds*, while *m* for minutes or *h* for *hours*, or a time of the form *mm:ss* giving minutes and seconds may be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

EXAMPLES

This example limits the size of all 'core' files generated to 0 bytes. This means that no 'core' files will be created if a program terminates abnormally.

```
limit coredumpsize 0
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type given in the message occurred.

CAVEATS

A child process may unlimit any resources limited by the parent.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh),

jobs(1csh), kill(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), ulimit(1sh), umask(1csh), unhash(1csh), unalias(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), brk(2), getrlimit(2), sbrk(2), setrlimit(2), ulimit(3c).

NAME

line — read one line

SYNOPSIS

line [*timeout*]

DESCRIPTION

Line copies one line (up to a newline) 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.

The *timeout* argument is an integer which specifies the maximum time in seconds to wait for input. If no input is given in that amount of time, **line** exits with a code of 1.

EXAMPLES

The following is a shell script that asks the user for a file name and prints the contents of the script. When the user types an end–of–file, the script exits.

```
#!/bin/sh
while true
do
        Name=`line`
        if [ $? != 0 ]
        then
                exit 0
        fi
        cat "$Name"
done
```

The following is a shell script that waits for the user to type something, reminding him that it is waiting every 10 seconds.

```
#!/bin/sh
while true
do
        if line 0
        then
                exit 0
        fi
        echo "\007Waiting for response"
done
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[1]	End-of-file reached or timeout with no input data.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

sh(1sh), read(2), select(2).

NAME

lint — a C program verifier

SYNOPSIS

lint [**—abchnpuvxz**] file . . .

DESCRIPTION

Lint attempts to detect features of the C program *files* which are likely to be bugs, or non–portable, or wasteful. It also checks the type usage of the program more strictly than the compilers. Among the things which are currently found 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.

By default, it is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to **lint**; these libraries are referred to by a conventional name, such as '—lm', in the style of *ld*(1). Arguments ending in *.ln* are also treated as library files. To create lint libraries, use the **—C** option:

lint —Cfoo files . . .

where *files* are the C sources of library *foo*. The result is a file *llib-lfoo.ln* in the correct library format suitable for linting programs using *foo*.

exit(3c) and other functions which do not return are not understood; this causes various lies.

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.

/*NOSTRICT*/
: shuts off strict type checking in the next expression.

/*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.

OPTIONS

Any number of the options in the following list may be used. The **—D**, **—U**, and **—I** options of *cc*(1) are also recognized as separate arguments.

—a Report assignments of long values to int variables.

—b Report *break* statements that cannot be reached. (This is not the default because, unfortunately, most *lex* and many *yacc* outputs produce dozens of such comments.)

—c Complain about casts which have questionable portability.

—h Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.

—n Do not check compatibility against the standard library.

—p Attempt to check portability to the *IBM* and *GCOS* dialects of C.

—u Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running **lint** on a subset of files out of a larger program).

—v Suppress complaints about unused arguments in functions.

—x Report variables referred to by extern declarations, but never used.

—z Do not complain about structures that are never defined (e.g. using a structure pointer without knowing its contents.).

FILES

/usr/lib/lint/lint[12]	programs
/usr/lib/lint/llib-lc.ln	declarations for standard functions
/usr/lib/lint/llib-lc	human readable version of above
/usr/lib/lint/llib-port.ln	declarations for portable functions
/usr/lib/lint/llib-port	human readable portable functions
llib-l.ln*	library created with **—C** option

CAVEATS

There are some things you just **can't** get **lint** to shut up about.

SEE ALSO

cc(1).

NAME

list — list vector

SYNOPSIS

list [**—c***n*] [**—d***string*] [*vector ...*]

DESCRIPTION

Output is a listing of the elements of the input *vector*(s). If no *vector* is given, the standard input is assumed.

OPTIONS

—c*n*

n is the number of output elements per line. The default value is 5.

—d*string*

The characters in *string* serve as delimiters. Only elements that are delimited by these characters will be listed. The "white space" characters space, tab, and linefeed (newline) are always delimiters.

EXAMPLES

The following example outputs each element of A, three per line.

```
list -c3 A
```

The following example outputs each element of A that is delimited by commas or "white space" characters, five per line. A comma requires two backslashes because it is a special character for **list**.

```
list -d\\, A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

ls, lf, lg, ll, lr, lx — list contents of directory

SYNOPSIS

ls [**—1**] [**—A**] [**—C**] [**—F**] [**—L**] [**—R**] [**—a**] [**—c**] [**—d**] [**—f**] [**—g**] [**—h**] [**—i**] [**—l**] [**—q**] [**—r**] [**—s**] [**—t**] [**—u**] [**—x**] *filename...*

DESCRIPTION

For each argument that names a directory or symbolic link to a directory (unless **—l** is specified), **ls** lists the contents of the directory; for each file argument, **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.

The commands **lf, lg, ll, lr,** and **lx** are links to **ls** which set flags as follows:

lf	–F
lg	–l, –g
ll	–l
lr	–R
lx	–x

The mode printed under the **—l** option contains 11 characters which are interpreted as follows: the first character is

b if the entry is a block–type special file;
c if the entry is a character–type special file;
d if the entry is a directory;
l if the entry is a symbolic link, or
— 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. 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 the set–group–id bit set; likewise the user–execute permission character is given as **s** if the file has the set–user–id bit set.

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 is printed. If you find the size using the **—s** option, this reflects the actual amount of disk space a file consumes. In this case, a count of indirect blocks is also printed. If you find the size using the **—l** option, a count of indirect blocks is NOT included. The size obtained with **—l** is not the actual number of bytes, but an end of data location. Unless the file was created by skipping around and writing, this is the actual number of bytes consumed.

OPTIONS

—a List all entries; in the absence of this option, entries whose names begin with a period (.) are *not* listed.

—A
list all entries except '.' and '..'. This option is always set for the super–user.

—c Use time of the last file status change for sorting or printing. This time is the last time that the file's data or the i–node was changed (meaning the last time the data was changed or the file was linked to. See the manual page for *stat(2)* for more information).

—C
force multi–column output; this is the default when output is to a terminal.

—d If argument is a directory, list only its name; often used with **—l** to get the status of a 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.

—F cause directories to be marked with a trailing '/', symbolic links with a trailing '@', and executable files with a trailing '*'. All symbolic links to existing directories are marked with a trailing '@/'. All symbolic links to existing executable files are marked with a trailing '@*'. All symbolic links to nonexistent files are marked with a trailing '@?'.

—g Include the group ownership of the file in a long output.

—h Follow only "hard" directory paths during recursion (**—R** option). By default, symbolic links to directories are traversed (except when **—l** is used without **—L**). This option prevents these symbolic links from being traversed.

—i For each file, print the i–number in the first column of the report.

—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. If the file is a symbolic link the pathname of the linked–to file is printed preceded by "–>".

—L If argument is a symbolic link, list the file or directory the link references rather than the link itself.

—q force printing of non–graphic characters in file names as the character '?'; this is the default when output is to a terminal.

—r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

—R
recursively list subdirectories encountered. (Note: In order to avoid symbolic link loops, directories are only traversed the first time they are encountered, whether as a regular directory or as a symbolic link.)

—s Give size in kilobytes of each file.

—t Sort by time modified (latest first) instead of by name.

—u Use time of last access instead of last modification for sorting (with the **—t** option) and/or printing (with the **—l** option).

—x List only directories. If the **—L** option is also given, symbolic links to directories will also be listed.

—1 force one entry per line output format; this is the default when output is not to a terminal.

EXAMPLES

The following example lists all of the files in the directory */bin*.

```
ls /bin
```

FILES

/etc/passwd	to get user id's for 'ls —l'.
/etc/group	to get group id's for 'ls —g'.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The date printed in the long listing (**—l** option) will contain the time of day if the file is up to six months old or has a date up to 48 hours in the future. All other dates will print with the year instead of the time.

The option setting based on whether the output is a teletype is undesirable as *ls —s* is much different than *ls —s ¦ lpr* . On the other hand, not doing this setting would make old shell scripts which used **ls** almost certain losers.

The commands *lf(1), lg(1), ll(1), lr(1),* and *lx(1)* are not separate programs; they are links to **ls.** This means that removing these commands will not free up a lot of disk space. This also means that modifying **ls** requires re–linking the commands.

SEE ALSO

cat(1), chmod(1), find(1), rm(1).

NAME

ln — make links

SYNOPSIS

ln [**—s**] [**—f**] name1 [name2]
ln [**—s**] [**—f**] name ... directory

DESCRIPTION

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc.) may have several links to it. There are two kinds of links: hard links and symbolic links.

By default **ln** makes hard links. A hard link to a file is indistinguishable from the original directory entry; any changes to a file are effective independent of the name used to reference the file. Hard links may not span file systems and may not refer to directories.

The **—s** option causes **ln** to create symbolic links. A symbolic link contains the name of the file to which it is linked. The referenced file is used when an *open*(2) or *creat*(2) operation is performed on the link. A *stat*(2) on a symbolic link will return the linked–to file; an *lstat*(2) must be done to obtain information about the link. The *readlink*(2) call may be used to read the contents of a symbolic link. Symbolic links may span file systems and may refer to directories.

The **—f** option causes **ln** to attempt to make hard links to directories. This can only be done by the super–user and can cause problems (see CAVEATS).

Given one or two arguments, **ln** creates a link to an existing file *name1*. If *name2* is given, the link has that name; *name2* may also be a directory in which to place the link; otherwise it is placed in the current directory. If only the directory is specified, the link will be made to the last component of *name1*.

Given more than two arguments, **ln** makes links to all the named files in the named directory. The links made will have the same name as the files being linked to.

OPTIONS

—f Option causes **ln** to attempt to make hard links to directories. This can only be done by the super–user and can cause problems (see CAVEATS).

—s **ln** creates a symbolic link.

EXAMPLES

The following example causes a hard link called c to be made to the file */bin/cat*.

```
ln /bin/cat c
```

This example causes a symbolic link called temp to be made to the directory */tmp*.

```
ln -s /tmp temp
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

The force option (–f) should only be used when absolutely neccessary. If a hard link is made to a directory and both the directory and link are removed, the system may crash.

SEE ALSO

rm(1), cp(1), mv(1), link(2), readlink(2), stat(2), open(2), creat(2), lstat(2), symlink(2).

NAME

log — logarithm

SYNOPSIS

log [**—b***n*] [**—c***n*] [*vector* ...]

DESCRIPTION

Output is the logarithm for each element of the input *vector(s)*. If no *vector* is given, the standard input is assumed.

OPTIONS

—b*n*

n is the logarithm base. If not given, *e* (2.71828...) is used.

—c*n*

n is the number of output elements per line.

EXAMPLES

The following example outputs the logarithm base 2 of each element of A, three per line.

```
log -b2,c3 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g)*.

NAME

login — sign on

SYNOPSIS

login [username]

DESCRIPTION

The **login** command is used at the beginning of each terminal session and allows the user to identify himself to the system. It may be invoked by the user 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 end–of–file.

If **login** is invoked without the argument *username*, it asks for a user name, and, if appropriate, a password. Echoing is turned off (if possible) during the typing of the password, so it will not appear on the written record of the session.

If a user does not complete the login successfully within a certain period of time (e.g., one minute), he is likely to be silently disconnected.

After a successful login, accounting files are updated, the user is informed of the existence of mail, and the message of the day and the time of the user's last login is printed. This output to the user is suppressed if a file named *.hushlogin* is in the user's home directory. (This feature is mostly used to make life easier for non–human users, such as *uucp*.)

Login initializes the user and group IDs and the working directory, then executes a command interpreter (usually *sh(1sh)* or *csh(1csh)*) according to specifications found in a password file. Argument 0 of the command interpreter is the name of the command interpreter with a leading dash ("—") prepended.

Login also initializes the basic environment (see *environ(7)*) with information specifying home directory, command interpreter, command search path, terminal type (if available), user name and mail directory. The environment is initialized to:

HOME = *your-login-directory*
SHELL = *last-field-of-passwd-entry*
PATH = :/bin:/usr/bin
TERM = *first-field-of-ttytype-entry*

If the file */etc/nologin* exists **login** prints its contents on the user's terminal and exits. This is used by *shutdown(8)* to stop users logging in when the system is about to go down.

Login is recognized by *sh(1sh)* and *csh(1csh)* and executed directly (without forking).

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting

/usr/adm/lastlog	accounting
*/usr/spool/mail/**	mail
/etc/motd	message–of–the–day
/etc/passwd	password file
/etc/ttytype	ttytype file
/etc/nologin	stops logins
.hushlogin	makes login quieter
/etc/securetty	lists ttys that root may log in on

DIAGNOSTICS

Login incorrect
The name or the password is bad.

No Shell No directory
The */etc/passwd* entry is not set up correctly for this user, and there may be no home directory as well. See a system administrator.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

An undocumented option, **—r** is used by the remote login server, *rlogind(8n)* to force **login** to enter into an initial connection protocol.

SEE ALSO

mail(1mh), passwd(1), passwd(5), environ(7), init(8), getty(8), shutdown(8).

NAME

logout — terminate login shell (**csh** built-in)

SYNOPSIS

logout

DESCRIPTION

The **logout** command terminates a login shell, causing the file $HOME/.logout to be read for commands to execute prior to logging out. This is especially useful when the variable *ignoreeof,* which causes ^D characters to not terminate the shell is set.

EXAMPLES

The following .logout file will cause the terminal screen to be cleared and the date to be printed before logging out.

```
clear
date
```

FILES

$HOME/.logout
File containing commands to be executed before logging out.

CAVEATS

If there are stopped jobs when **logout** is executed, a message is printed and the **logout** is aborted. If executed again, all stopped jobs are sent the continue and terminate signals in order to abort them. All running jobs are allowed to continue.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), login(1), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

look — find lines in a sorted list

SYNOPSIS

look [**—d**] [**—f**] string [*filename*]

DESCRIPTION

Look consults a sorted *file* and prints all lines that begin with *string*. It uses binary search.

If no *filename* is specified, */usr/dict/words* is assumed with collating sequence **—df.**

OPTIONS

—d 'Dictionary' order: only letters, digits, tabs and blanks participate in comparisons.

—f Fold. Upper case letters compare equal to lower case.

EXAMPLES

The following invocation of this command will print all words beginning with 'exp' found in /usr/dict/words. Dictionary order and folding are automatically assumed.

```
look exp
```

FILES

/usr/dict/words	Online dictionary

RETURN VALUE

[0]	*String* was found.
[1]	*String* was not found.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

awk(1), comm(1), cut(1), egrep(1), fgrep(1), grep(1), paste(1), sort(1), spell(1), uniq(1).

NAME

lpq — print queue status via MDQS

SYNOPSIS

lpq [**—q** *queue*]

DESCRIPTION

Lpq calls the MDQS program *qstat(1mdqs),* selecting printer queue information. See *qstat* for detailed information.

Lpq returns request identifications for queued print jobs, where a request identification is in the form *user.job#*. The job number is all that is usually required for *lprm*.

OPTIONS

—q *queue*

specifies queue configured in */etc/qconf*. Default queue is *printqueue* specified in */etc/qconf*. *Qstat* **—a** lists all available queues.

EXAMPLES

lpq

FILES

/etc/qconf	MDQS configuration file

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

Rsh privileges are required in order to display requests in a remote queue.

Things can change while **lpq** is running; the picture it gives is only a close approximation of reality. For instance, **lpq** may produce false error messages if it cannot find a particular file or if a data structure it is looking at changes underneath it.

SEE ALSO

qstat(1mdqs), lpr(1mdqs), lprm(1mdqs).

NAME

lpr — queued printing via MDQS

SYNOPSIS

lpr [**—a** *atime*] [**—b**] [**—c** *count*] [**—f** *form*] [**—h** *headerfile*] [**—H** *Headerfile*] [**—l** *limit*] [**—m**] [**—n**] [**—p** *priority*] [**—q** *queue*] [**—r**] [**—t** *title*] [**—u** *useraddr*] [**—v**] [**—R** *ruser*] [*filename...*]

DESCRIPTION

Lpr causes the named files to be queued for printing in a lineprinter queue. If no files are specified, the standard input is read until an end of file. The original file(s) can be modified or deleted and the file(s) printed will be the file(s) as of the time of request submission. If the **—n** or **—r** option is used, the original file(s) are not spooled, hence the original file(s) should not be modified or deleted until printing of the file(s) has completed.

Once the request has been queued, it can be deleted using the *lprm* program or modified using the *qmod* program, and the current status determined using the *lpq* program.

A print request may be directed to either a local or a remote printer.

OPTIONS

—a *atime*
: do not start this job until after *atime*. See *getdate(5mdqs)* for a full description of time specifications. The most useful time specification is *hour:minute [meridian]*; if no meridian – am or pm – is specified, a 24–hour clock is used.

—b Print a banner page only for the first file. Normally a banner page is printed preceeding each printed file.

—c *count*
: print *count* copies of the specified files.

—f *form*
: print the files on the specified form. The default form is specified in */etc/qconf*. *Qdev* can be executed to determine appropriate forms.

—h *headerfile*
: use the file *headerfile* for banner page logo information instead of the system default. The full pathname of *headerfile* must be specified. The first ~20 lines of this file will be printed. The rest of the page is reserved for system provided information.

—H *Headerfile*
: use the file *Headerfile* for banner page logo information instead of the system default. *Headerfile* should be specified by a simple path name and should reside in the *print-hdrdir* directory specified in */etc/qconf*.

—l *limit* causes the print request to be limited to *limit* pages. The default limit is specified in */etc/qconf*. The *limit* should be in the range

0–32767. A *limit* of 0 indicates no limit.

—m cause a message to be generated upon the completion of the print request and sent via *mail(1mh)* to the requesting user.

—n will cause **lpr** not to spool the specified files, but to instead enter the filename only. This option may be used when files to be printed are extremely long and available space for second copies is limited. When using this option and directing the request to a remote printer, local spooling is bypassed but remote spooling is always done.

—p *priority*
allows one to give a request a priority other than the default. The priority is in the range 0–10, with 5 the default value. Only the superuser, the mdqs user or a member of the systems group may request a priority of 0.

—q *queue*
allows one to queue a request to a queue other than the default. Queueing a request to other than a print queue will give unpredictable results. The default print queue is specified in */etc/qconf*.

—r will cause **lpr** to remove the specified files upon successful completion of the request. If the request is to be printed on a remote host the files will not be removed. The **—r** option implies the **—n** option.

—R *ruser*
When a request is to print on a remote host MDQS normally requires the requesting local user name to exist on the remote host with *rsh(1n)* privileges. This option allows one to specify another user on the remote host for *rsh* privileges.

—t *title* overrides the default header page title with the string *title*.

—u *useraddr*
overrides the default notification address with the address *useraddr*. This address should be a valid address to your mailsystem. The default notification address is the name of the invoker on the current host.

—v Normally lpr does its work silently. The v (verbose) option makes lpr acknowledge that the request has been queued and prints the request number.

EXAMPLES

```
lpr file1 file2
```

Prints two files to the default printer with added pagination only between files.

```
pr file1 ¦ lpr
```

Paginates and prints one file to the wide printer.

```
lpr file1
```

qconf Queue/Device Mapping Description entry:

```
lp lp0 /usr/lib/mdqs/lpserver
```

Shows the qconf entry for a file to be printed on the current host.

```
lpr -q lp2 file1
```

qconf Queue/Device Mapping Description entry:

```
lp2 net /usr/lib/mdqs/netsend jondoe lp
```

(qconf file at jondoe contains Queue–Device Mapping and Device Description for queue "lp2")

Shows the qconf entry for a file to be rerouted to a remote host (jondoe) and printed there.

```
lpr -c 3 -h headerfile file1
```

Prints three copies of a file with the first 40 lines of headerfile included on the banner page.

FILES

/etc/qconf	configuration information for MDQS
/usr/lib/mdqs/forms	restriction list of available forms
/usr/lib/mdqs/lpserver	program called by the daemon to print files
/usr/lib/mdqs/netsend	program called to send to remote queue
/usr/lib/mdqs/queue2	second stage queuer, called by **lpr**
/usr/spool/q	top of spooling directory tree

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

When specifing **—n** (no copy), remote requests are always spooled.

SEE ALSO

lpq(1mdqs), lprm(1mdqs), qstat(1mdqs), qmod(1mdqs), forms(5mdqs), mdqsd(8mdqs), batch(1mdqs), rsh(1n).

NAME

lprm — queued print request removal via MDQS

SYNOPSIS

lprm [**—q** *queue*] *[user#.]job# ...*

DESCRIPTION

Lprm calls *qmod(1mdqs),* to remove a request from the print queue. See *qmod(1mdqs)* for detailed information.

The request identification obtained from *lpq(1mdqs)* is in the form *user.job#*. Normally only the *job#* is required for **lprm.**

If the queue specified is directed to a remote device, **lprm** will attempt to remove the job on the remote machine. If a request is directed to a remote device, but the request remains in the local network queue, **qmod** can be executed to remove or redirect the request.

OPTIONS

—q *queue*

specifies queue configured in */etc/qconf*. Default queue is *print-queue* specified in */etc/qconf*.

EXAMPLES

The following will remove a print job from the print queue, where the request identification gleaned from *lpq(1mdqs)* was, for example, jondoe.5.

```
lprm 5
```

FILES

/etc/qconf	MDQS configuration file

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

Rsh privileges are required for removing requests from remote queues. **Lprm** will not remove active jobs; see *qdev(8mdqs)* for flushing active requests.

SEE ALSO

qmod(1mdqs), lpq(1mdqs), lpr(1mdqs), qstat(1mdqs), qdev(8mdqs).

NAME

ls, lf, lg, ll, lr, lx — list contents of directory

SYNOPSIS

ls [**—1**] [**—A**] [**—C**] [**—F**] [**—L**] [**—R**] [**—a**] [**—c**] [**—d**] [**—f**] [**—g**] [**—h**] [**—i**] [**—l**] [**—q**] [**—r**] [**—s**] [**—t**] [**—u**] [**—x**] *filename...*

DESCRIPTION

For each argument that names a directory or symbolic link to a directory (unless **—l** is specified), **ls** lists the contents of the directory; for each file argument, **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.

The commands **lf, lg, ll, lr,** and **lx** are links to **ls** which set flags as follows:

lf	–F
lg	–l, –g
ll	–l
lr	–R
lx	–x

The mode printed under the **—l** option contains 11 characters which are interpreted as follows: the first character is

b if the entry is a block–type special file;
c if the entry is a character–type special file;
d if the entry is a directory;
l if the entry is a symbolic link, or
— 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. 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 the set–group–id bit set; likewise the user–execute permission character is given as **s** if the file has the set–user–id bit set.

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 is printed. If you find the size using the **—s** option, this reflects the actual amount of disk space a file consumes. In this case, a count of indirect blocks is also printed. If you find the size using the **—l** option, a count of indirect blocks is NOT included. The size obtained with **—l** is not the actual number of bytes, but an end of data location. Unless the file was created by skipping around and writing, this is the actual number of bytes consumed.

OPTIONS

—a List all entries; in the absence of this option, entries whose names begin with a period (**.**) are *not* listed.

—A
list all entries except '.' and '..'. This option is always set for the super–user.

—c Use time of the last file status change for sorting or printing. This time is the last time that the file's data or the i–node was changed (meaning the last time the data was changed or the file was linked to. See the manual page for *stat(2)* for more information).

—C
force multi–column output; this is the default when output is to a terminal.

—d If argument is a directory, list only its name; often used with **—l** to get the status of a 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.

—F cause directories to be marked with a trailing '/', symbolic links with a trailing '@', and executable files with a trailing '*'. All symbolic links to existing directories are marked with a trailing '@/'. All symbolic links to existing executable files are marked with a trailing '@*'. All symbolic links to nonexistent files are marked with a trailing '@?'.

—g Include the group ownership of the file in a long output.

—h Follow only "hard" directory paths during recursion (**—R** option). By default, symbolic links to directories are traversed (except when **—l** is used without **—L**). This option prevents these symbolic links from being traversed.

—i For each file, print the i–number in the first column of the report.

—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. If the file is a symbolic link the pathname of the linked–to file is printed preceded by "–>".

—L If argument is a symbolic link, list the file or directory the link references rather than the link itself.

—q force printing of non–graphic characters in file names as the character '?'; this is the default when output is to a terminal.

—r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

—R
recursively list subdirectories encountered. (Note: In order to avoid symbolic link loops, directories are only traversed the first time they are encountered, whether as a regular directory or as a symbolic link.)

—s Give size in kilobytes of each file.

—t Sort by time modified (latest first) instead of by name.

—u Use time of last access instead of last modification for sorting (with the **—t** option) and/or printing (with the **—l** option).

—x List only directories. If the **—L** option is also given, symbolic links to directories will also be listed.

—1 force one entry per line output format; this is the default when output is not to a terminal.

EXAMPLES

The following example lists all of the files in the directory */bin*.

```
ls /bin
```

FILES

/etc/passwd	to get user id's for 'ls —l'.
/etc/group	to get group id's for 'ls —g'.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The date printed in the long listing (**—l** option) will contain the time of day if the file is up to six months old or has a date up to 48 hours in the future. All other dates will print with the year instead of the time.

The option setting based on whether the output is a teletype is undesirable as *ls —s* is much different than *ls —s ¦ lpr* . On the other hand, not doing this setting would make old shell scripts which used **ls** almost certain losers.

The commands *lf(1), lg(1), ll(1), lr(1),* and *lx(1)* are not separate programs; they are links to **ls.** This means that removing these commands will not free up a lot of disk space. This also means that modifying **ls** requires re–linking the commands.

SEE ALSO

cat(1), chmod(1), find(1), rm(1).

NAME

lreg — linear regression

SYNOPSIS

lreg [**—F***vector*] [**—i**] [**—o**] [**—s**] [*vector ...*]

DESCRIPTION

Output is the slope and intercept from a least squares linear regression of each *vector* on a *base* vector. The *base* vector is specified using the **–F** option. If the *base* is not given, it is assumed to be ascending positive integers from zero.

OPTIONS

—F*vector*
vector is the *base*.

—i Only output the intercept.

—o Output the slope and intercept in "option" form (see *siline(1g)*).

—s Only output the slope.

EXAMPLES

The following example outputs the intercept from the linear regression of *vector* B on *base* vector A.

```
lreg -FA,i B
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g)*.

NAME

ls, lf, lg, ll, lr, lx — list contents of directory

SYNOPSIS

ls [**—1**] [**—A**] [**—C**] [**—F**] [**—L**] [**—R**] [**—a**] [**—c**] [**—d**] [**—f**] [**—g**] [**—h**] [**—i**] [**—l**] [**—q**] [**—r**] [**—s**] [**—t**] [**—u**] [**—x**] *filename...*

DESCRIPTION

For each argument that names a directory or symbolic link to a directory (unless **—l** is specified), **ls** lists the contents of the directory; for each file argument, **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.

The commands **lf, lg, ll, lr,** and **lx** are links to **ls** which set flags as follows:

lf	–F
lg	–l, –g
ll	–l
lr	–R
lx	–x

The mode printed under the **—l** option contains 11 characters which are interpreted as follows: the first character is

b if the entry is a block–type special file;
c if the entry is a character–type special file;
d if the entry is a directory;
l if the entry is a symbolic link, or
— 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. 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 the set–group–id bit set; likewise the user–execute permission character is given as **s** if the file has the set–user–id bit set.

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 is printed. If you find the size using the **—s** option, this reflects the actual amount of disk space a file consumes. In this case, a count of indirect blocks is also printed. If you find the size using the **—l** option, a count of indirect blocks is NOT included. The size obtained with **—l** is not the actual number of bytes, but an end of data location. Unless the file was created by skipping around and writing, this is the actual number of bytes consumed.

OPTIONS

—a List all entries; in the absence of this option, entries whose names begin with a period (.) are *not* listed.

—A
list all entries except '.' and '..'. This option is always set for the super–user.

—c Use time of the last file status change for sorting or printing. This time is the last time that the file's data or the i–node was changed (meaning the last time the data was changed or the file was linked to. See the manual page for *stat(2)* for more information).

—C
force multi–column output; this is the default when output is to a terminal.

—d If argument is a directory, list only its name; often used with **—l** to get the status of a 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.

—F cause directories to be marked with a trailing '/', symbolic links with a trailing '@', and executable files with a trailing '*'. All symbolic links to existing directories are marked with a trailing '@/'. All symbolic links to existing executable files are marked with a trailing '@*'. All symbolic links to nonexistent files are marked with a trailing '@?'.

—g Include the group ownership of the file in a long output.

—h Follow only "hard" directory paths during recursion (**—R** option). By default, symbolic links to directories are traversed (except when **—l** is used without **—L**). This option prevents these symbolic links from being traversed.

—i For each file, print the i–number in the first column of the report.

—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. If the file is a symbolic link the pathname of the linked–to file is printed preceded by "–>".

—L If argument is a symbolic link, list the file or directory the link references rather than the link itself.

—q force printing of non–graphic characters in file names as the character '?'; this is the default when output is to a terminal.

—r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

—R
recursively list subdirectories encountered. (Note: In order to avoid symbolic link loops, directories are only traversed the first time they are encountered, whether as a regular directory or as a symbolic link.)

—s Give size in kilobytes of each file.

—t Sort by time modified (latest first) instead of by name.

—u Use time of last access instead of last modification for sorting (with the **—t** option) and/or printing (with the **—l** option).

—x List only directories. If the **—L** option is also given, symbolic links to directories will also be listed.

—1 force one entry per line output format; this is the default when output is not to a terminal.

EXAMPLES

The following example lists all of the files in the directory */bin*.

```
ls /bin
```

FILES

/etc/passwd	to get user id's for 'ls —l'.
/etc/group	to get group id's for 'ls —g'.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The date printed in the long listing (**—l** option) will contain the time of day if the file is up to six months old or has a date up to 48 hours in the future. All other dates will print with the year instead of the time.

The option setting based on whether the output is a teletype is undesirable as *ls —s* is much different than *ls —s ¦ lpr* . On the other hand, not doing this setting would make old shell scripts which used **ls** almost certain losers.

The commands *lf(1), lg(1), ll(1), lr(1),* and *lx(1)* are not separate programs; they are links to **ls.** This means that removing these commands will not free up a lot of disk space. This also means that modifying **ls** requires re–linking the commands.

SEE ALSO

cat(1), chmod(1), find(1), rm(1).

NAME

lsh — limited shell

SYNOPSIS

lsh [**—acefhiknrstuvx**] [arg] ...

DESCRIPTION

Lsh is a limited version of *sh(1sh).*

While using **lsh,** the user may not use the commands *cd* or *chdir,* reset the shell variables PATH and SHELL, or specify any command name containing a '/'.

Upon starting a login shell, commands are read from the file */etc/lsh.profile* if it exists, and then from *$HOME/.profile.*

The system administrator should set up these files so that they are not writeable by the user (and should therefore be owned by root). The PATH variable should contain only directories containing "safe" commands. It is recommended that these commands be placed in */usr/lbin* and only commands which do not allow the user to start up a normal shell.

Options, examples, and other information may be found in the manual page for *sh(1sh).*

FILES

$HOME/.profile	This file is read and commands contained in it executed when the shell is called as '–lsh', usually at login.
*/tmp/sh**	Temporary storage for storing arguments to '<<'.
/etc/lsh.profile	This file is read and commands contained in it executed when the shell is called as 'lsh'.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2).

NAME

ls, lf, lg, ll, lr, lx — list contents of directory

SYNOPSIS

ls [**—1**] [**—A**] [**—C**] [**—F**] [**—L**] [**—R**] [**—a**] [**—c**] [**—d**] [**—f**] [**—g**] [**—h**] [**—i**] [**—l**] [**—q**] [**—r**] [**—s**] [**—t**] [**—u**] [**—x**] *filename...*

DESCRIPTION

For each argument that names a directory or symbolic link to a directory (unless **—l** is specified), **ls** lists the contents of the directory; for each file argument, **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.

The commands **lf, lg, ll, lr,** and **lx** are links to **ls** which set flags as follows:

lf –F

lg –l, –g

ll –l

lr –R

lx –x

The mode printed under the **—l** option contains 11 characters which are interpreted as follows: the first character is

b if the entry is a block–type special file;
c if the entry is a character–type special file;
d if the entry is a directory;
l if the entry is a symbolic link, or
— 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. 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 the set–group–id bit set; likewise the user–execute permission character is given as **s** if the file has the set–user–id bit set.

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 is printed. If you find the size using the **—s** option, this reflects the actual amount of disk space a file consumes. In this case, a count of indirect blocks is also printed. If you find the size using the **—l** option, a count of indirect blocks is NOT included. The size obtained with **—l** is not the actual number of bytes, but an end of data location. Unless the file was created by skipping around and writing, this is the actual number of bytes consumed.

OPTIONS

—a List all entries; in the absence of this option, entries whose names begin with a period (**.**) are *not* listed.

—A
list all entries except '.' and '..'. This option is always set for the super–user.

—c Use time of the last file status change for sorting or printing. This time is the last time that the file's data or the i–node was changed (meaning the last time the data was changed or the file was linked to. See the manual page for *stat(2)* for more information).

—C
force multi–column output; this is the default when output is to a terminal.

—d If argument is a directory, list only its name; often used with **—l** to get the status of a 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.

—F cause directories to be marked with a trailing '/', symbolic links with a trailing '@', and executable files with a trailing '*'. All symbolic links to existing directories are marked with a trailing '@/'. All symbolic links to existing executable files are marked with a trailing '@*'. All symbolic links to nonexistent files are marked with a trailing '@?'.

—g Include the group ownership of the file in a long output.

—h Follow only "hard" directory paths during recursion (**—R** option). By default, symbolic links to directories are traversed (except when **—l** is used without **—L**). This option prevents these symbolic links from being traversed.

—i For each file, print the i–number in the first column of the report.

—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. If the file is a symbolic link the pathname of the linked–to file is printed preceded by "–>".

—L If argument is a symbolic link, list the file or directory the link references rather than the link itself.

—q force printing of non–graphic characters in file names as the character '?'; this is the default when output is to a terminal.

—r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

—R
recursively list subdirectories encountered. (Note: In order to avoid symbolic link loops, directories are only traversed the first time they are encountered, whether as a regular directory or as a symbolic link.)

—s Give size in kilobytes of each file.

—t Sort by time modified (latest first) instead of by name.

—u Use time of last access instead of last modification for sorting (with the **—t** option) and/or printing (with the **—l** option).

—x List only directories. If the **—L** option is also given, symbolic links to directories will also be listed.

—1 force one entry per line output format; this is the default when output is not to a terminal.

EXAMPLES

The following example lists all of the files in the directory */bin*.

```
ls /bin
```

FILES

/etc/passwd	to get user id's for 'ls —l'.
/etc/group	to get group id's for 'ls —g'.

RETURN VALUE

[NO_ERRS] Command completed without error.

[NP_ERR] An error occurred that was not a system error. Execution terminated.

[P_WARN] A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The date printed in the long listing (**—l** option) will contain the time of day if the file is up to six months old or has a date up to 48 hours in the future. All other dates will print with the year instead of the time.

The option setting based on whether the output is a teletype is undesirable as *ls —s* is much different than *ls —s ¦ lpr* . On the other hand, not doing this setting would make old shell scripts which used **ls** almost certain losers.

The commands *lf(1), lg(1), ll(1), lr(1),* and *lx(1)* are not separate programs; they are links to **ls.** This means that removing these commands will not free up a lot of disk space. This also means that modifying **ls** requires re–linking the commands.

SEE ALSO

cat(1), chmod(1), find(1), rm(1).

NAME

m4 — macro processor

SYNOPSIS

M4 [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 arguments, or if an argument is '—', the standard input is read. The processed text is written on the standard output.

Macro calls have the form

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

The '(' must immediately follow the name of the macro. If a defined macro name is not followed by a '(', it is deemed to have no arguments. Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore '_', where the first character is not a digit.

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. 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.

undefine
removes the definition of the macro named in its argument.

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 UTek versions of **m4**.

changequote
Change quote characters to the first and second arguments. *Changequote* without arguments restores the original values (i.e., ` ').

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 newline.

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

incr
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.

eval
evaluates its argument as an arithmetic expression, using 32–bit arithmetic. Operators include +, —, *, /, %, ^ (exponentiation); relationals; parentheses.

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 UN*X command given in the first argument. No value is returned.

maketemp
fills in a string of XXXXX in its argument with the current process id.

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.

SEE ALSO

ratfor(1).

NAME

mail — read or send MH mail

SYNOPSIS

mail [*username...*] [**—cc** *username...*] [**—body** *text*] [**—subject** *text*]

DESCRIPTION

If **mail** is called with no arguments, it calls *inc* to incorporate mail from the user's incoming mail drop */usr/spool/mail/$USER)* into the user's *inbox* folder. The new messages being incorporated are assigned numbers starting with the next highest number in the folder. If the *inbox* folder doesn't exist, the user will be queried prior to its creation. As the messages are processed, a *scan* listing of the new mail is produced. See *inc*(1mh) for further details.

If any arguments are passed to **mail**, it attempts to send mail to the userids listed. Names listed after the **—cc** switch will be entered in the 'Cc:' component of the message. Names listed before the **—cc** switch (or if the switch is omitted) are added to the 'To:' component of the message.

The string following the **—subject** switch forms the 'Subject:' component of the message. If the switch is omitted, there will be no 'Subject:' component in the message.

If the **—body** switch is present, the string that follows it will form the body of the message being constructed. If the switch is omitted, the body will be taken from standard input.

The primary purpose of the **mail** program is to provide an interface that is compatible with the old UNIX mail system so that programs that wish to send mail may still do so.

Your *.mh_profile* can contain the following entries:

Path	To determine the user's MH directory
Folder—Protect	For protection on new folders
Msg—Protect	For protection on new messages

OPTIONS

—body *text*
The text following the **—body** switch will form the body of the message being constructed.

—cc *name*
Names listed after the **—cc** switch will be entered in the "Cc:" of the message.

—subject *text*
The text following the **—subject** switch will be entered in the "Subject:" component of the message.

FILES

$HOME/.mh_profile	The user profile
/usr/spool/mail/$USER	The user's mail drop

CAVEATS

Mail refuses to send mail messages of length zero.

SEE ALSO

comp(1mh), fmt(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), mh(1mh), mhl(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh), mh(5mh), mh_profile(5mh).

NAME

make — build files and programs

SYNOPSIS

make [**—B**] [**—N**] [**—R**] [**—S**] [**—Y**] [**—d**] [**—e**]
[**—f** *makefile*] [**—g**] [**—i**] [**—k**] [**—l**] [**—m**] [**—n**] [**—p**]
[**—q**] [**—r**] [**—s**] [**—t**] [*macro definitions*] [*target ...*]

DESCRIPTION

Make is a program for maintaining programs. It looks at the last modification date of files to determine the actions it is to perform in order to make sure that the *target* is up to date. This document briefly describes the syntax of the rules and how they work. A more detailed description is available in the document *Using Make.*

Rules

Make uses special rules in order to decide what commands it must execute. There are two types of rules that make uses: target rules and suffix rules.

Target rules have the following form :

target : [:] [*dependents*] ⟨*separator*⟩ [*commands*]

The *target* and the *dependents* may each be one or more valid file names, except that spaces and tabs are not allowed. Target names beginning with '.' are treated specially and should not be used, since **make** thinks that they are special entries. The *separator* may either be a semicolon, or a newline and a tab. The rule itself specifies that the files in *target* may be made up to date by executing the *commands.* The *dependents* are the names of the files or targets that must be up to date before the *commands* are executed.

The double colon (::) has a special meaning to **make.** There may be more than one rule for targets if this is used, and each rule is used in order. The first rule for the target must be finished before all others are checked. This is especially useful for maintaining archive files (again, see the document *Using Make).*

Suffix rules have the following form :

⟨*suffix1*⟩⟨*suffix2*⟩: ⟨*separator*⟩ [*commands*]

suffix1 is a non–null file name suffix, like **.c** or **,v** which may begin with a period, but may not contain any other periods. *suffix2* may be null (see CAVEATS), but must begin with a period if it is not. The *separator* may either be a semicolon or a newline and a tab. See the next section for the description of command syntax.

The rule itself specifies that if a file ending with *suffix1* exists and has the same root (everything but the suffix) as a file or target name ending with *suffix2,* then the given *commands* will make the latter file up to date.

Special Characters in Rules
The dependency line has some constraints. The characters ':', '>', '&', '|', space, and tab may not appear in file names unless the characters are preceded by the character '\'. Because of this added ability to have special characters in file names, all default suffix rules put file names in quotes. It is a good idea to do this with suffix rules defined in the makefile as well. See the document *Using Make* for more information on this subject.

Command Syntax
The commands mentioned in the above section are commands executed by *sh(1sh),* unless the special entry .CSHELL is placed in the makefile. Before executing the commands, all words beginning with the character '$' are expanded as macros (see the Macros section). The commands are then executed by *sh.* If the command exits with a nonzero status, **make** will normally stop. This can be turned off globally by the **—i** flag or the **.IGNORE** entry (see the OPTIONS and Special Entries sections). Nonzero exit status can be ignored for a single command by preceding the command with the character '–'.

Normally, **make** prints each command before it is executed. The **—s** option and the **.SILENT** entry turn this off globally. Commands preceded by the character '@' are not printed.

In order to speed things up, **make** looks at the command line after macros have been expanded to see whether it can directly execute the command, instead of having the shell execute it. This means that shell commands like *times* and *cd* will be executed directly by **make** so that the shell built–in command is not executed. In most cases, this is not a problem, since each command line is executed separately. For example, the target rule:

```
target :
        cd ..
        make all
```

will cause the error message

```
make : cd : No such file or directory
```

to be printed. This error message is misleading, because the real error is that the second line should be `cd ..;\` so that **make** executes the command 'sh –ce cd ..;make all'. It is not worth the effort to have **make** recognize all of the built–in shell commands.

Rules Files
Make reads rules from a file called a 'makefile'. The default makefiles are named *makefile* and *Makefile.* The **—f** option can be used to specify an alternate file.

The first step **make** takes is to make sure that the default makefiles are up to date if they are being used. In order to do this, **make** has a special set of built–in rules. Also, the file */usr/lib/mfrules* is read for system–wide

rules. If the file *$HOME/.mfilerc* exists, it is read for additional rules. The environment variable MFRULES can be set to the name of a file to use instead of *$HOME/.mfilerc*. Because of this action, makefiles can be stored under RCS and do not have to be present when **make** is invoked. This action can be turned off with the **—m** option and is not performed if the **—f** option is given. The **—n** and **—N** flags are ignored during this step, so the makefile is always made up to date.

There is a set of built–in rules which **make** uses. Because of these rules, there does not have to be a makefile available. In addition to these rules, the files */usr/lib/makerules* and *$HOME/.makerc* (which can be changed by setting the MAKERULES environment variable) are read. The use of each of these files can be selectively turned off by the **—r, —B, —Y,** and **—R** options (see the OPTIONS section). The complete set of default rules and macro definitions is available by executing the command:

```
make -pfn /dev/null 2>/dev/null
```

(The 2>/dev/null sends the error message that is printed due to the absence of a target to /dev/null.)

Invocation

Make needs to know what it is supposed to be making. If there is no makefile, the target must be specified on the command line. In this case, only implicit rules are used. This is useful for creating programs from a single source file. If there is a makefile, the target may be specified on the command line. If no target is specified, the first target name found in the makefile that does not begin with a period is used. Many makefiles have a target rule for 'all' as the first target rule. This target usually depends on the program or programs that the makefile is being used for. In this way, executing **make** with no arguments implies **make** *all*.

More than one target may be specified. Each target is made up to date in the order they are given. If an error occurs while making a target, **make** stops, unless the **–k** option is given.

Options are first taken from the command line and then from the environment variable MAKEFLAGS. If the **—e** option is given on the command line, the MAKEFLAGS variable is ignored. The flags **—e** and **—f** are not allowed in the MAKEFLAGS variable. The variable may contain spaces and '–' characters for readability, but these are ignored by **make.** All options given to **make** except for the **—f** are stored in the special macro **MFLAGS.** Because of this, the makefile can contain a command like:

```
$(MAKE) $(MFLAGS) clean
```

and flags given to the top level command will be given to lower level invocations. The maximum number of options which may be given to **make** is 100. If more are given, an error message is printed and execution terminates.

Directory Searching
Make always looks in the current directory for files. The special entry **.DIRECTORIES** contains a list of other directories to look in for files. This is especially useful when source files are kept in an RCS directory or when programs share source. The default list of directories contains only ./RCS. This, coupled with the default suffix rule for .c,v.c, means that the file *file.c* implicitly depends on the file *./RCS/file.c,v*.

Directory searching is not done in cases where explicit dependencies are specified or with names containing the slash character (/) or beginning with the character '.'.

Macros
Make has a form of macro substitution. A macro definition is of the form "NAME = text". These may be specified in any rules file or on the command line. Any occurence of $(NAME) or ${NAME} found in the makefile is expanded to the text. Single letter macro names do not need the parentheses or braces. If a macro is defined multiply in a file, the last definition is the only one that is stored.

Make also has a set of special macros. These are:

Macro
Expands to

**$$** the character '$'

$MAKE
the name of the make program being executed (this is normally **make.**

$MFLAGS
the flags given to this invocation of **make**

$@ the full name of the current target

$< the name of the file which caused the implicit rule to be used

$? the list of dependencies which were out of date (explicit rules only)

$* the root of the target name

$% the name of the member of the current archive file (only set for archive dependencies)

There are three modifiers for the last five special macros given. The form **$(***macro***D)** expands to the directory names of each element of the expanded macro, if there is one. The form **$(***macro***F)** expands to the basename of the file names in the expanded macro. The form **$(***macro***:***from=to***)** expands to the names of each element in the macro, replacing occurrences of the *from* text which appear at the end of words with the *to* string. For example, if $? has the value *file1.o file2.o file3.o* , $(?:.o = .c) will expand to *file1.c file2.c file3.c*.

Special Entries
There are some special entries which can be placed in the makefile to set options and specify data. The form of these entries is NAME : [data]. These entries are :

Name
Result

.IGNORE
Nonzero exit statuses are ignored. (Same as the **—i** option.)

.SILENT
Supresses printing of commands. (Same as the **—s** option.)

.DEBUG
Print debugging output. (Same as the **—d** option.)

.GRAPH
Print a dependency list graph. (Same as the **—g** option.)

.CONTINUE
On nonzero status, stop processing the current target, but continue with targets that do not depend on the current target. (Same as the **—k** option.)

.PRINT
Print information about commands, targets, macros, and other makefile entries. (Same as the **—p** option.)

.DEFAULT
When a target can not be made up to date or made to exist, the commands described by the .DEFAULT entry are executed. A typical use of this entry is to get SCCS files, since this version of make does not support SCCS files.

.PRECIOUS
Don't delete the file being built if fatal error occurs. This entry requires a list of filenames as data.

.SUFFIXES
With no data, the suffix list is set to null. With a list of suffixes as data, the suffixes are added to the current suffix list. Unless the **—I** option is given, the special suffix .NULL is replaced in the list by the null string. This is used in suffix rules where the second suffix is null.

.DIRECTORIES
With no data, the directory list is set to null. With a list of directories as data, the directories are added to the directory list.

.CSHELL
Normally, commands that must be executed by a shell are executed by */bin/sh*. If this entry is in the makefile, commands are executed by */bin/csh* with the **—f** option (meaning that the file .cshrc is not

read). This also causes the character '~' to be special. This should only be used by experts.

Makefile Formats
The makefiles may contain comments as well as rules and macro definitions. The character '#' denotes that the rest of the current line is a comment.

All words separated by spaces are taken to be distinct words. For this reason, the entry "my file", which is a valid filename to *sh* is separated into the word **my** and the word " **file**" instead of being treated as a single name.

If a line ends with a backslash (\), the newline and all tabs, spaces, and newlines following it are ignored. This is useful for maintaining very long macros and for formatting commands for readability. This can also be dangerous, since this feature is not turned off inside of comments.

OPTIONS

—B
Do not read the built–in default rules.

—N
Trace and print, but do not execute the commands needed to update the targets, unless the word '$(MAKE)' or '${MAKE}' appears in the command line. This enables the user to trace multi–level makefiles.

—R
Do not read the personal default rules file.

—S
Stop after any command fails.

—Y
Do not read the system default rules file.

—d Print debugging output. See the document *Using Make* for an explanation of the output.

—e Do not read the MAKEFLAGS environment variable.

—f *filename*
Specifies a different makefile than *makefile* or *Makefile*

—g Print a dependency list. The dependencies for each file are indented over four spaces.

—i Ignore nonzero exit status. This is equivalent to the special entry .IGNORE.

—k When a command returns nonzero status, abandon work on the current entry, but continue on branches that do not depend on the current entry.

—l Remove the null suffix from the suffix list.

—m Do not try to make the makefiles up to date.

—n Trace and print, but do not execute the commands needed to update the targets.

—p Print a listing containing information about all rules, macros, special entries and dependencies.

—q If the target is up to date, exit with status 0. Otherwise, exit with status —1.

—r Don't use the built–in rules and don't read the system or personal rules files. This is equivalent to the flags **—BYR**

—s Equivalent to the special entry '.SILENT:'.

—t Touch, i.e. update the modified date of targets, without executing any commands.

EXAMPLES

This example builds a program called 'put' from the source file

```
make put
```

This example makes the program 'get' from the files 'get.c' and 'subs.c', both of which require the include file 'subs.h'. The makefile looks like:

```
get : get.o subs.o
        $(CC) $(CFLAGS) -o $@ get.o subs.o

get.o subs.o : subs.h
```

The invocation:

```
make get
```

will cause *get.c* and *subs.c* to be compiled to create the object files. If either of these files, *subs.h*, or the makefile are stored under RCS and not present in the current directory, they will be checked out. Finally, the two object files will be compiled together to create the program *get*.

This example reads rules from the file *make.pgm* and makes the programs *docput*and *dprint*. After finishing, it installs the programs and cleans up.

```
make -f make.pgm docput dprint install clean
```

(Typically, the *install* target moves the programs to the proper directory for execution, and the *clean* target removes all object files, temporary files, and sources, if the sources are maintained under RCS.)

This example prints out a graph of the dependencies used by make when the command 'make' with no arguments is executed.

```
make -mnsg
```

Note the use of the **—m** to supress information about making the makefiles, **—n** so no commands are executed, and **—s** to supress printing of commands to be executed.

This example builds the programs 'docput' and 'dprint' as above, but if errors occur in building 'docput', processing is still done on

```
make -k docput dprint
```

FILES

[mM]akefile	The file *makefile* is used as the rules file if it exists. If not, the file *Makefile* is used if it exists.
/usr/lib/mfrules	Additions to the default rule set for making makefiles. Provided for users with no source code.
$HOME/.mfilerc	Personal makefile-making rules.
/usr/lib/makerules	Additions to the default rule set. Provided for users with no source code.
$HOME/.makerc	Personal default rules.

VARIABLES

HOME	The user's home directory.
MAKEFLAGS	Flags to **make**.
MFRULES	The name of the personal makefile-making rules file, if *$HOME/.mfilerc* is not used."
MAKERULES	The name of the personal default rules file, if *$HOME/.makerc* is not used."

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[-1]	When the **—q** option is given, a return value of -1 means that the target is not up to date.

[NP_WARN] An error warranting a warning message occurred. Execution continues.

[NP_ERR] An error occurred that was not a system error. Execution terminated.

[P_WARN] A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

[P_ERR] A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Some commands return nonzero status inappropriately. Use **—i** to overcome the difficulty.

Commands that are directly executed by the shell, notably *cd*(1sh), are ineffectual across newlines in **make.**

When using single–suffix rules (where the second suffix is null), the target may have no explicit dependencies or commands.

There are cases in which makefiles which work with other versions of **make** which do not handle RCS directories will have problems with this version. This is almost always the result of incompletely specified dependents. See the EXAMPLES section for an example of a rule which has this property.

By default, there exists on a rule for building archives (files with the suffix ".a") for C files (with the suffix ".c"). When using other types of files to build archives, rules are required.

The **—t** flag should be used with care. In cases where one or more of the targets processed is not supposed to be a file (as in a makefile with a mnemonic *print* target), an empty file will be created.

When a line ends with a backslash, all spaces, tabs, and newlines on the following lines are ignored. For example, this block of text

```
prog : prog.c\

prog2 : prog2.c
        prog prog2.c ¦ cc -c -o prog2

prog3 : prog3.c
```

is interpreted as

```
prog : prog.c prog2 : prog2.c
        prog prog2.c ¦ cc -c -o prog2

prog3 : prog3.c
```

which will result in a syntax error.

When macro substitution causes a line in the makefile to expand to longer than 2500 characters, a memory fault may occur. Checking is done to try to prevent this, but it is not possible to guarantee that this gets caught before the damage is done.

SEE ALSO

cc(1), co(1rcs), f77(1), pc(1), sh(1sh), touch(1).

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 (that is, 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, upper– and lower–case letters, and '.' and '/'. 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 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 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 useful key bits in the result.

Makekey is intended for programs that perform encryption (for instance, *ed* and *crypt*(1)). Usually **makekey**'s input and output will be pipes.

RETURN VALUE

[NO_ERRS] Command completed without error.

SEE ALSO

crypt(1), ed(1), ex(1), crypt(3c).

NAME

makewhatis — build whatis databases

SYNOPSIS

/usr/lib/makewhatis [**—v**] *directory* ...

DESCRIPTION

Makehwhatis builds whatis database files as described in *whatis(5man)*. For each of the named directories, **makewhatis** builds a file named 'whatis' from the formatted manual pages in the subdirectories 'cat[1–8]', and places it in that directory.

Each database entry is built from the NAME section of the formatted page. All bold and underlined characters are processed to remove the backspaces and multiple characters, resulting in an entry of normal text. Newlines and multiple tabs and spaces are compressed into single spaces. A valid NAME section will be of the following form:

command1 [, command2, ...] – description

One database entry is created for each 'command' listed in the NAME section (see *whatis(5man)*). Only the first 2047 characters (after compression and removal of bold and underlined sequences) of the description are used.

Invalid NAME sections are ignored and no messages are printed unless the **—v** option is given.

When the NAME section contains more than one 'command' name, a link is created for each command to the original manual page. For example, if the manual page /usr/man/cat3/setuid.3c lists the commands 'setuid' and 'getuid', a link will be made to /usr/man/cat3/setuid.3c called /usr/man/cat3/getuid.3c.

OPTIONS

—v Verbose. For each subdirectory processed, a message is printed. Also, any invalid NAME sections or links that can not be made result in an error message.

EXAMPLES

The following invocation builds the files '/usr/man/whatis' and '/usr/man/manl/whatis'.

```
makewhatis /usr/man /usr/man/manl
```

This invocation builds the file '/usr/man/whatis' and prints out the name of each subdirectory being processed and notes any problems found.

```
makewhatis -v /usr/man
```

FILES

whatis	The name of the whatis database file.

VARIABLES

HOME	The user's home directory.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

The resulting whatis database is sorted by section number only. No sorting is done within the sections, so multiple copies of the same manual page will result in multiple entries in the database.

SEE ALSO

apropos(1man), buildif(1man), help(1man), man(1man), section(1man), whatis(1man), man(5man), manindex(5man), whatis(5man), catman(8man).

NAME

man — find and print manual entries

SYNOPSIS

man [**—a**] [**—b** *separator*] [**—i** *ignoresecs*] [**—n**] [**—o**] [**—r**] [**—s**] [*section*] *title ...*
man [**—k** *(apropos(1man) options)*]
man [**—f** *(whatis(1man) options)*]

DESCRIPTION

When called with the **—k** option, the utility **apropos** is executed with the remaining arguments.

When called with the **—f** option, the utility **whatis** is executed with the remaining arguments.

Normally, **man** searches a set of directories named in the file */usr/lib/man/directories* for a formatted manual page entry file and either prints the name or the file contents.

Each manual page file has a name of the form 'title.section'. A 'section' is a number followed by zero or more alphabetic characters. For example, this document is contained in the file 'man.1'. The 'title' to be searched for is given on the command line. An ordered list of known 'section' names is found in the file */usr/lib/man/sections*.

By default, all known sections are searched for manual entries. If the *section* argument is given, only that section or set of sections is printed. If the *section* argument is of the form '#+', all sections that begin with the number specified by '#' are searched. If the **—i** option is given, the sections in *ignoresecs* are ignored. The *ignoresecs* argument is a list of section names separated by commas, spaces, and/or tabs.

The default search method takes each section and searches for matching pages in all directories. The alternate method, which takes each directory and searches for matching pages in all sections, is available by specifying the **—s** option.

By default, if the standard output is a terminal, the output is piped through *more(1)*. If the standard output is not a terminal, the output is printed with no processing. All pages are printed together, separated by a separator, which is described later. This line serves to separate the pages on the lineprinter or within **more**, where it can also be used to search for the beginning of the next page in a series.

Man searches for manual pages in subdirectories of the directories specified in */usr/lib/man/directories*. Each entry in this file has a corresponding command directory. The environment variable PATH is used to order the manual page directories. This means that if */bin* is in the search path before */usr/local* , manual pages corresponding to the commands in */bin* are searched before those corresponding to */usr/local*. In order to tailor the actions of the **man** command, the file *$HOME/.manrc* may be set up. This file may contain one of each of the following entries, except for *personal:*, of which there may be up to 20:

options: *options* — where *options* is a list of command line options not including **—f, —k,** and **—i.**

ignore: *sections* — where *sections* is a list of section names to ignore separated by commas, spaces, and/or tabs. This entry is ignored if the **—i** option is given or if a section to search is specified on the command line.

output: *command* — where *command* is a command line which is run by the shell. The command must be able to read from the standard input.

personal: *directory* — where *directory* is the name of a directory which contains subdirectories as described in *man(5man)*. These directories are searched in the order they appear in the manrc file for manual pages before any other directories. If *directory* begins with the sequence *$HOME/* , the *$HOME* is replaced by the value of the *HOME* environment variable.

sections: *sections* — where *sections* is an ordered list of sections separated by commas, spaces, and/or tabs. This list may include '+' sections as well as specific sections. For example, the list "1sh, 1+, 2+, 3+, 3f, 4+, 5+, 7+, 8+" specifies that section '1sh' is to be searched before any other sections beginning with a 1, and section '3f' is to be searched after all other sections beginning with 3.

tty-sep: *separator* — where *separator* is text to be printed between manual page entries when the output is going to a terminal. The text is taken from the first non-whitespace character to the end of the line. The following special escaped characters are processed as in *echo(1sh):* \n (newline), \f (formfeed), \r (carriage return), \b (backspace), (tab), \\ (backslash), and \c (inhibit final newline). Unless the line contains a \c or ends with a backslash (except for \\), a newline is added to the separator.

notty-sep: *separator* where *separator* is text to be printed between manual page entries when the output is not going to a terminal.

OPTIONS

—a Print information for all manual pages. Normally, only the first page is used.

—b *separator*
Print the given separator between manual page entries. Backslashes in the separator string are processed as described above under the $HOME/.manrc "tty-sep" entry. This option overrides the "tty-sep" and "notty-sep" entries in the $HOME/.manrc file. An empty separator causes nothing to be printed between manual entries.

—f Execute the command **whatis** with the remaining arguments.

—i *ignoresecs*
Ignore the sections in the *ignoresecs* list. This option overrides the

—k Execute the command **apropos** with the remaining arguments.

—n Print only the pathnames of the files containing the matching pages.

—o Print output with no processing even if standard output is a terminal.

—r Ignore the "options:" entry in *$HOME/.manrc*.

—s Use the alternate search method (by directory).

EXAMPLES

The following invocation will print the manual page for the subroutine *exit* in any of the subsections of section 3, if it exists.

```
man 3+ exit
```

FILES

/usr/lib/man/sections	Known manual page sections.
/usr/lib/man/directories	Manual page search directory information.
$HOME/.manrc	**Man** command control information.

VARIABLES

PATH	The user's execution path.
HOME	The user's home directory.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.

CAVEATS

There is no way to specify which directories to search other than in the PATH variable.

Since output is piped to another process, the message "Broken pipe" may appear if the output command is terminated before the entire manual page is processed.

Man is insensitive to the case of section names. For example, section '1c' is the same as '1C'.

If the **—k** option or the **—f** option is used, that option must be the first option. Options appearing before the **—k** or **—f** will be ignored, with the exception of arguments that are invalid to **man.**

The **man** command knows how to avoid the index format data (if there is any) at the bottom of the formatted manual page entry, so it is best to always use **man** to access manual page entries.

SEE ALSO

apropos(1man), buildif(1man), echo(1sh), help(1man), makewhatis(1man), manintro(1man), more(1), section(1man), whatis(1man), man(5man), manindex(5man), whatis(5man), catman(8man).

NAME

manintro — introduction to the online manual page system

DESCRIPTION

The online manual page system allows a user to read entries in the user's manual and personal manual pages stored on the system.

Manual page formats

Each manual page contains a number of sections, each of which gives some kind of information about the subject, such as how a command or subroutine works. The first section is alwats the NAME section, which gives the names of the subjects being described and a short description. Other sections discuss the subject and give examples and related documents.

Initial installation

When the system is first installed, the directory /usr/man will contain the subdirectories "man[1–8]" and "cat[1–8]" which correspond to the major manual page sections from the printed user's manual. The 'cat' directories, which will contain the formatted manual pages will be empty. The sections contain the following types of information:

1 Commands and command set introductions

2 System calls

3 Library subroutines

4 Special system files and hardware support

5 System file formats

6 Games (usually nonexistent)

7 Document processing macros and special concepts

8 System administration commands

Once the manual page sources are installed in the 'man' directories, the program must be run to format all of the pages. This command may be run automatically each night by adding an entry to the file /usr/lib/crontab (see *catman(8man)* for details).

If you have your own manual pages that you would like to use with the system, see the manual page *man(5man)* for more information.

If you do not plan to change the manual pages, you can delete the contents of the 'man' directories after **catman** has formatted them all (make sure you put them somewhere, such as on a backup disk, in case something happens to the formatted pages). This will save disk space and will make the nightly **catman** runs go faster. For the most part, it isn't useful to keep the sources around, since the manual page commands do not require them.

Dirstributed manual pages: If you have more than one system and are running the distributed file system, you may want to choose one machine on which to store the manual pages. You can access these pages from

other machines by setting up the file /usr/lib/man/directories to contain the names of these directories. For example, if you have two machines called "jimshost" and "janeshost" and you have the manual pages in /usr/man on the machine "jimshost", users on the machine "janeshost" can access the pages if the entries

```
//jimshost/usr/man      /bin
//jimshost/usr/man      /etc
```

are added to /usr/lib/man/directories. See *man(5man)* for information about the contents of this file.

Commands

There are a number of commands which make up the online manual page system. Following is a brief description of each of the commands. More information is available in the manual pages for the commands.

/bin/man	This command is the interface to the actual manual pages. It allows reading of pages on the terminal and other facilities related to the formatted manual pages.
/bin/apropos	This command is an interface to a set of special database files which contain short descriptions of all of the manual pages. It allows searching for manual pages based on keywords.
/bin/whatis	This command is another interface to the same files used by *apropos(1man)*. It allows one to get short descriptions of manual pages by giving the subject name.
/bin/help	This command is an interactive manual page peruser. You can read certain sections of manual pages and look around at other pages without leaving the peruser.
/bin/section	This command is a non–interactive version of *help(1man)*. You can display sections of manual pages separately.
/etc/catman	This command formats the manual pages and executes the next two commands.

/usr/lib/makewhatis	This command builds the special database used by **apropos** and **whatis,** and creates links to manual pages which describe more than one subject.
/usr/lib/buildif	This command builds special format information tables for use by **help** and **section.**

Special features

This manual page system contains features which allow users to decide which sets of pages they want to see by default, set up and maintain personal manual page directories. Also, this system makes information easier to get, since each command, subroutine, and special subject described in the manual has a manual entry.

SEE ALSO

apropos(1man), buildif(1man), help(1man), makewhatis(1man), man(1man), section(1man), whatis(1man), man(5man), manindex(5man), whatis(5man), catman(8man).

NAME

mean — arithmetic mean

SYNOPSIS

mean [—**f***n*] [—**p***n*] [—**n***n*] [*vector ...*]

DESCRIPTION

Output is the mean of the elements in the input *vector(s)*. The input may optionally be trimmed. If no *vector* is given, the standard input is assumed.

OPTIONS

—**f***n*

Trim (1/*n*)*r elements from each end, where r is the rank of the input vector.

—**p***n*

Trim *n**r elements from each end, where *n* is between 0 and .5.

—**n***n*

Trim *n* elements from each end.

EXAMPLES

The following example outputs the mean of the middle 50% of the elements of A, i.e., A is trimmed by 25% of its elements from both ends.

```
mean -p.25 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

merge — three–way file merge

SYNOPSIS

merge [**—p**] file1 file2 file3

DESCRIPTION

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

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

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

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

OPTIONS

—p merge sends its output to std. output.

EXAMPLES

The following command incorporates into *example.c* the differences between the files *old.example.c* and *newer.example.c*.

```
merge example.c old.example.c newer.example.c
```

FILES

/tmp/d3[abc]$$	Temporary storage for file pair differences.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the error message occurred.

SEE ALSO

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

NAME

mesg — permit or deny messages

SYNOPSIS

mesg [[–]{**ny**}]

DESCRIPTION

Mesg with argument **n** or **—n** forbids messages via *write*(1) by revoking non–user write permission on the user's terminal. **Mesg** with argument **y** or **—y** reinstates permission. **Mesg** with no arguments reports the current state of standard input (stdin) without changing it.

OPTIONS

—n Forbid messages.

n Forbid messages.

—y Allow messages.

y Allow messages.

EXAMPLES

The following invocation will tell the user what the message permission currently is on their tty.

```
mesg
```

FILES

*/dev/tty**

RETURN VALUE

[0]	messages are receivable
[1]	messages are not receivable
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

write(1).

NAME

mh — Rand Message Handler

DESCRIPTION

MH is the name of the Rand Message Handling system. Rather than being a single comprehensive program, **MH** consists of a collection of fairly simple single–purpose programs to send, receive, save, and retrieve messages.

SEE ALSO

comp(1mh), fmt(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), mh(1mh), mhl(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh), mh(5mh), mh_profile(5mh).

NAME

mhl — produce formatted listings of MH messages

SYNOPSIS

mhl [**—clear**] [**—noclear**] [**—folder** folder] [**—form** formfile] [**—length** length] [**—width** width] [**—help**] [*filename* . . .]

DESCRIPTION

Mhl is a formatted message listing program. It can be used as a replacement for *more(1)* (the default message lister). The default lister can be changed by placing a "showproc: /bin/mhl" in your ".mh_profile". As with *more(1),* each of the messages specified as arguments (or stdin) will be output. If more than one message file is specified, the user will be prompted prior to each one, and a <RETURN> or <eot> will begin the output, with <RETURN> clearing the screen (if appropriate), and <eot> suppressing the screen clear. An <interrupt> will abort the current message ouput, prompting for the next message (if there is one), and a <quit> will terminate the program (without core dump).

As in all MH programs, **mhl** looks for a line *mhl: args* in the user's profile, and thus allows tailored defaults. The switches are:

—clear	Clear screen each page
—noclear	Don't clear screen each page
—folder*folder*	Use this "folder" name
—form*formfile*	Name of the format file
—length*length*	Screen length
—width*width*	Screen width
—help	Standard help message

All of the functions these switches perform are affected or controlled by information elsewhere. That is, the format file can specify "clear", in which case, the command line switches will override. Also, the length and width can be specified in the format file, or default to 40x80. The folder is used in constructing a message name (see special component "MessageName" below). If it is not specified in a switch, it is taken from the environment variable *mhfolder* , which *show*, *next*, *prev*, and *pick*initialize appropriately.

If the form file is not specified, it is taken from the file *mhl.format* in the user's MH directory; if that file doesn't exist, it is taken from */usr/lib/mh/mhl.format*. (Same evaluation hierarchy as the compose form file.)

Mhl operates in two phases: 1) read and parse the format file, and 2) process each message (file). During phase 1, an internal description of the format is produced as a structured list. In phase 2, this list is walked for each message, outputting message information under the format constraints from the format file.

The *mhl.format* file contains information controlling screen clearing, screen size, wraparound control, transparent text, component ordering, and component formatting. Also, a list of components to ignore may be specified, and a couple of "special" components are defined to provide added functionality. Message output will be in the order specified by the order in the format file.

Each line of *mhl.format* has one of the formats:

```
;comment
:cleartext
variable[,variable . . . ]
component:[variable, . . . ]
```

A line beginning with a ";" is a comment, and is ignored. A line beginning with a ":" is clear text, and is output exactly as is. (A line containing only a ":" produces a blank line in the output.) A line beginning with "component:" defines the format for the specified component, and finally, remaining lines define the global environment.

For example, the line:

width = 80,length = 40,clearscreen,overflowtext = ***,overflowoffset = 5

defines the screen size to be 80 columns by 40 rows, specifies that the screen should be cleared prior to each page, that the overflow indentation is 5, and that overflow text should be flagged with "***".

Following are all of the current variables and their arguments. If they follow a component, they apply only to that component, otherwise, their effect is global. Since the whole format is parsed before any output processing, the last global switch setting for a variable applies to the whole message.

width = #	Screen width, component width
overflowtext = t	Text to use at the beginning of an overflow line
overflowoffset = #	Positions to indent overflow lines
compwidth = #	Positions to indent component text after first line
nocomponent	Don't output "component: " for this component
uppercase	Output text of this component in all upper case
center	Center component on line (works for one–line components only)
clearscreen	Clear the screen (form feed) prior to each page
leftadjust	Strip off leading spaces & tabs on each line of text
compress	Change newlines in text to spaces

Where "=#" indicates a number must be specified, and "=t" indicates that arbitrary text up to end of line or "," is required. The variables without arguments are ON indicators, with the default in all cases OFF. The variables "nocomponent", center, leftadjust and compress have no affect globally, and clearscreen only affects the global environment.

A line of the form:

ignores=component,...

specifies a list of components which are never output.

The component "MessageName" (case is unimportant) will output the actual message name (*filename*) preceded by *folder:* if one is specified or found in the environment.

The component "Extras" will output all of the components of the message which were not matched by explicit components, or included in the ignore list. If this component is not specified, an ignore list is not needed since all non-specified components will be ignored.

If "nocomponent" is NOT specified, then the component name will be output as it appears in the format file.

The current default format is:

```
width=80,length=40,overflowtext=***,overflowoffset=5
ignores=msgid,message-id
Date:leftadjust,offset=40
To:leftadjust
Cc:leftadjust
:
From:leftadjust
Subject:leftadjust
:
extras:leftadjust,nocomponent
:
body:nocomponent
```

OPTIONS

—clear	Clear the screen for each page.
—folder *folder*	Use this "folder" name.
—form *formfile*	Name of the format file.
—help	Standard help message.
—length *length*	Set screen length.
—noclear	Don't clear the screen for each page.
—width *width*	Set screen width.

SEE ALSO

inc(1mh), more(1), next(1mh), prev(1mh), show(1mh), mh_profile(5mh).

NAME

mhpath — print full pathnames of MH messages and folders

SYNOPSIS

mhpath [+**folder**] [**msgs**] [—**help**]

DESCRIPTION

Mhpath expands and sorts the message list [msgs] and writes the full pathnames of the messages to the standard output separated by newlines. If no "msgs" are specified, **mhpath** outputs the folder (directory) pathname (current folder default). Contrasted with other MH commands, a message argument to **mhpath** may often be intended for *writing*. Because of this: 1) the name *new* has been added to **mhpath**'s list of reserved message names (the others are *first*, *last*, *prev*, *next*, *cur*, *all*). *New* is last + 1 (where last is 0 in a messageless folder). *New* may not be used as part of a message range. 2) Within a message list, the following designations may refer to messages that do not exist: a single numeric message name, the single message name cur, and (obviously) the single message name *new*. All other message designations must refer to at least one existing message. 3) An empty folder is not in itself an error. Message numbers greater than 999 as part of a range designation are replaced with 999. Explicit single message numbers greater than 999, or message number 0 in any context, are errors.

Your .mh_profile can contain the following entries:

Path	To determine the user's MH directory
Current-Folder	To find the default current folder

mhpath has the following defaults:

+folder defaults to current
msgs defaults to NULL

The folder and the current message are unaffected.

OPTIONS

—help
Display a synopsis of the **mhpath** command.

+folder
folder becomes the current folder.

EXAMPLES

The current folder foo contains messages 3 5 6. Cur is 4.

```
% mhpath
/r/phyl/Mail/foo
% mhpath all
/r/phyl/Mail/foo/3
/r/phyl/Mail/foo/5
/r/phyl/Mail/foo/6
% mhpath 1000
```

```
Message 1000 out of range 1-999
% mhpath 1-1001
/r/phyl/Mail/foo/3
/r/phyl/Mail/foo/5
/r/phyl/Mail/foo/6
% mhpath new
/r/phyl/Mail/foo/7
% mhpath last new
/r/phyl/Mail/foo/6
/r/phyl/Mail/foo/7

% mhpath last-new
Bad message list "last-new".

% mhpath cur
/r/phyl/Mail/foo/4

% mhpath 1-2
No messages in range "1-2".

% mhpath first:2
/r/phyl/Mail/foo/3
/r/phyl/Mail/foo/5
% mhpath 1 2
/r/phyl/Mail/foo/1
/r/phyl/Mail/foo/2

% mhpath 0
Bad message list "0".

% mhpath 0-last
Bad message list "0-last".
```

--Backquoted Operations--

```
% cd `mhpath +inbox`
From "e":
<CMD> run cat 'mhpath cur'
```

FILES

$HOME/.mh_profile The user profile

CAVEATS

Like all MH commands, **mhpath** expands and sorts [msgs]. Don't expect

```
mv mhpath 501 500
```

to move 501 to 500-- the reverse will happen. But

```
mv mhpath 501 mhpath 500'
```

will do the trick. Out of range message 0 is treated far more severely than large out of range message numbers.

SEE ALSO

mh(5mh).

NAME

mkdir — make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories whose modes are the exclusive–or of the user's umask and 777. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

EXAMPLES

The following example creates the directory Schedules that is readable and executable by everyone and writeable by the owner and group.

```
umask 002
mkdir Schedules
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

rmdir(1), sh(1sh), umask(1sh), umask(1csh), mkdir(2), umask(2).

NAME

mod — modulo function

SYNOPSIS

mod [—**c***n*] [—**m***n*] [*vector* ...]

DESCRIPTION

Output is a vector with each element being the remainder of dividing the corresponding element from the input *vector(s)* by the *modulus*. If no *vector* is given, the standard input is assumed.

OPTIONS

—**c***n*

n is the number of output elements per line.

—**m***n*

n is the *modulus*. If not given, 2 is used.

EXAMPLES

The following example outputs the elements of A modulo 8, three per line.

```
mod -m8,c3 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

more, page — file perusal filter for crt viewing

SYNOPSIS

more [**—c**] [**—d**] [**—f**] [**—l**] [**—p**] [**—s**] [**—u**] [*—n*] [*+linenumber*] [*+/pattern*] [*filename...*]

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a soft–copy 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. Other possibilities are enumerated later.

If the program is invoked as **page,** or with the **—p** option, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and k — 1 rather than k — 2 lines are printed in each screenful, where k is the number of lines the terminal can display.

More looks in the termcap entry (see TERMCAP in the VARIABLES section) to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre—set any flags desired. For example, if you prefer to view files using the *—c* mode of operation, the *sh* command sequence *MORE = '—c' ; export MORE* would cause all invocations of **more** , including invocations by programs such as *man* and *msgs* , to use this mode. The *MORE* environment variable is only checked for flags, so the preceding — is not required. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.profile* file.

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

Other sequences which may be typed when **more** pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i⟨space⟩
: display *i* more lines, (or another screenful if no argument is given)

i⟨cr⟩
: display *i* more lines, (or one line if no argument is given). The value of *i* becomes the new window size.

^D
: display 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d
: same as ^D (control—D)

*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

q or Q Exit from **more.**

= Display the current line number.

v Start up the editor *vi* at the current line.

h or ? Help command; give a description of all the **more** commands.

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.

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

' (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 characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n skip to the *i*—th next file given in the command line (skips to last file if i 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.

:f display the current file name and line number.

:q or :Q exit from **more** (same as q or Q).

. (dot) repeat the previous 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 (see *stty(1)*) 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).

A sample usage of **more** in previewing *nroff* output would be

```
nroff -ms +2 doc.n ¦ more -s
```

OPTIONS

—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 **More** will prompt the user with the message "Hit space to continue, Q or q to quit" at the end of each screenful. This is useful if **more** is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

—f This 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 Do not treat ^L (form feed) specially. If this option is not given, **more** will pause after any line that contains a ^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.

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

—p **More** will erase the entire screen before displaying the next page. This is the same as executing **page.**

—s Squeeze multiple empty lines from the output, producing only one empty line (lines with whitespace are not empty). Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

—u Normally, **more** will handle underlining and bold characters 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, bold or stand—out mode for underlined information in the source file. The *—u* option suppresses this processing.

+linenumber
Start up at *linenumber.*

+/pattern
Start up three lines before the line containing the regular expression *pattern.*

EXAMPLES

The following example will display the file text.c starting at line 65.

```
more +65 text.c
```

FILES

/etc/termcap	Default terminal data base.
/usr/lib/more.help	Help file.

VARIABLES

MORE	The options to be used when invoking more.
SHELL	The user's login shell. Used for shell escapes.
TERM	The type of terminal being used.
TERMCAP	The name of the file containing the terminal capability entry, or the entry itself.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

If no *ho* string exists in the termcap entry, cursor motion is used to home the cursor when the **-c** or **-p** options are used. In these cases, the *ti* string is printed before text is displayed, and the *te* string is printed before **more** exits. The manual page for *termcap(5t)* describes *ti* and *te* as the

strings required to enter/exit programs that use cursor motion. This means that *te* should not clear the screen! If it does, your screen will be cleared when you exit **more.**

Lines longer than 1024 characters are separated into 1024 character lines separated by newlines. The 1024 character limit applies to input characters, so a line that contains a large number of backspaces may be separated even if it prints as less than 1024 characters.

When **more** is run on files that are being written or appended, the percentage of the file viewed may be listed as more than 100%.

Numbers in the options are scanned sequentially and do not get reset. For example, the commands

```
more -1f1
more -11f
```

are equivalent.

When a line contains a pagefeed (^L), the entire line is displayed at the end of the page, instead of being split up.

SEE ALSO

man(1man), nroff(1), page(1), sh(1sh), stty(1), ul(1), termcap(5t), environ(7).

NAME

msghlp, sccshelp — ask for help about error messages

SYNOPSIS

msghlp args
sccshelp args

DESCRIPTION

Msghlp finds information to explain a message from a command or explain the use of a command. One or more arguments may be supplied.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, or 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., **sys13**, for message 13 from the *sys* command). The file */usr/lib/errtags/helploc* will be checked for a file corresponding to the non numeric prefix. That file will then be searched for the message. If */usr/lib/errtags/helploc* does not exist or the prefix is not found there the search will be attempted on */usr/lib/errtags/*⟨*non-numeric prefix*⟩, (e.g, */usr/lib/errtags/zz*), with the search key being ⟨remainder of arg⟩, (e.g., 32).

type 2 — Does not contain numerics (as a command, such as **get**) The file */usr/lib/errtags/cmds* is searched, with the search key being the whole argument.

type 3 — Is all numeric (e.g., **212**), or if the file as determined above does not exist, the search will be attempted on */usr/lib/errtags/default* with the search key being the entire argument.

The response of the program will be the explanatory information related to the argument, if there is any.

FILES

/usr/lib/errtags	directory containing files of message text.
/usr/lib/errtags/⟨*non-numeric prefix*⟩	the file searched if the message is not found in */usr/lib/errtags/helploc.*
/usr/lib/errtags/default	the file searched if the argument is all numeric.
/usr/lib/errtags/helploc	the file searched if the argument has a non-numeric prefix.
/usr/lib/errtags/cmds	contains the syntax lines for the commands.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.

SEE ALSO

sccshelp(1sccs), ERROR(3c), errtag(5).

NAME

sccshelp, msghlp — ask for help about error messages

SYNOPSIS

sccshelp *args*
msghlp *args*

DESCRIPTION

Sccshelp finds information to explain a message from a command or explain the use of a command. One or more arguments may be supplied.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, or one of the following types:

type 1 — Begins with nonnumerics, ends in numerics. The nonnumeric prefix is usually an abbreviation for the program or set of routines which produced the message (for example, **sys13**, for message 13 from the **sys** command). The file */usr/lib/errtags/helploc* will be checked for a file corresponding to the nonnumeric prefix. That file will then be searched for the message. If */usr/lib/errtags/helploc* does not exist or the prefix is not found there, the search will be attempted on */usr/lib/errtags/<non-numeric prefix>*, (for example, */usr/lib/errtags/zz)*, with the search key being *<remainder of arg>* (for example, 32).

type 2 — Does not contain numerics (as a command, such as **get**) The file */usr/lib/errtags/cmds* is searched, with the search key being the whole argument.

type 3 — Is all numeric (for example, **212**), or if the file as determined above does not exist, the search will be attempted on /usr/lib/errtags/default with the search key being the entire argument.

The response of the program will be the explanatory information related to the argument, if there is any.

EXAMPLES

This request is of type 2 described above. **Sccshelp** will search the file */usr/lib/errtags/cmds* to find information that will explain the command **ident:**

```
sccshelp ident
```

FILES

/usr/lib/errtags — Directory containing files of message text.

/usr/lib/errtags/<non-numeric prefix> — The file searched if the message is not found in */usr/lib/errtags/helploc.*

/usr/lib/errtags/default	The file searched if the argument is all numeric.
/usr/lib/errtags/helploc	The file searched if the argument has a nonnumeric prefix.
/usr/lib/errtags/cmds	Contains the syntax lines for the commands.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.

SEE ALSO

msghlp(1), ERROR(3c), errtag(5).

NAME

mt — magnetic tape manipulating program for 6100 Series workstations

SYNOPSIS

mt [**—f** *tapename*] *command* [*count*]

DESCRIPTION

Mt is used to give commands to a magnetic tape drive. If a tape name is not specified, the environment variable TAPE is used; if TAPE does not exist, **mt** uses the device **/dev/ntc**. Note that *tapename* must refer to a raw (not block) tape device and should usually refer to a no–rewind device.

By default **mt** performs the requested operation once. Operations may be performed multiple times by specifying *count*.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

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

fsf Forward space *count* files.

bsf Back space *count* files.

fsr Forward space *count* records.

bsr Back space *count* records.

rewind Rewind the tape (*Count* is ignored.)

offline, rewoffl
Rewind the tape and place the tape unit off–line (*Count* is ignored.)

status Print status information about the tape unit.

fss Forward space until *count* sequential file marks are found.

bss Back space until *count* sequential file marks are found.

erase Erase from the current position to the end of the tape. Some drivers will only allow this command at BOT. (*Count* is ignored.)

end Forward space to the end of recorded data. (*Count* is ignored.)

tension, retension
Retension cartridge tapes, leaving the tape at BOT. (*Count* is ignored.)

FILES

*/dev/ntc** cartridge tape interface

DIAGNOSTICS

command *command* not valid
Command wasn't one of the commands listed above or wasn't a unique abbreviation.

count must be positive

Count cannot be a negative number.

can't open *tapename*

Mt couldn't open the named tape. The device doesn't exist, the tape wasn't mounted, or the tape was write protected against a **weof** or **erase** command.

Other messages describe tape errors.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

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

NAME

mv — move or rename files

SYNOPSIS

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

mv [—] [**—f**] [**—i**] *filename...* directory

DESCRIPTION

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

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

Mv refuses to move a file onto itself.

OPTIONS

— means interpret all the following arguments to **mv** as file names. This allows file names starting with minus.

—f stands for force. Normally, when you try to move to a destination file that is not writeable, you are prompted as to whether or not to override this protection. When the **mv** is forced, you are not prompted at all, and the file movement takes place.

—i stands for interactive mode. Whenever a move is to overwrite an existing file, the user is prompted on standard output by the name of the file followed by a question mark. If on standard input, the user answers with a line starting with 'y', the move continues. Any other reply prevents the move from occurring.

EXAMPLES

The following example will move the file *a.out* to the directory *bin*.

```
mv a.out bin
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

If *file1* and *file2* lie on different file systems, **mv** must copy the file and delete the original. In this case the owner name becomes that of the

copying process and any linking relationship with other files is lost.

The **—i** option for **mv** is not the same as that for *rm*.

When a directory is moved via a command like *mv dir1 dir2* , the destination directory must be empty or nonexistent.

If the destination file is writeable but busy (being executed), **mv** will ask for the protection to be overridden.

SEE ALSO

cp(1), ln(1).

NAME

neqn — format mathematical text for nroff

SYNOPSIS

neqn [—**d***xy*] [—**p***n*] [—**s***n*] [—**f***n*] [files]

DESCRIPTION

Neqn is an *nroff*(1) preprocessor for typesetting mathematical text on typewriter–like terminals, while *eqn*(1) is used for the same purpose with *troff*(1) on a phototypesetter. Usage is almost always:

```
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: $\lim_{n \leftarrow \infty} \sum_0^n x_i$ 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 $\ddot{\overline{y}} = \underline{n}$, and *x vec ~ = ~ y dyad* is $\vec{x} = \vec{\vec{y}}$.

Point sizes and fonts can be changed with **size** *n* or **size** $\pm n$, **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** ($\sum$), **int** ($\int$), **inf** (∞), and shorthands such as >= ($\geq$), != ($\neq$), and —> ($\leftarrow$) 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. *Nroff(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 *nroff(1)* when all else fails. Full details are given in the manual cited below.

OPTIONS

—d*xy*	Sets the equation delimiters to *x* and *y*.
—f*n*	Changes globally, the font of the text.
—p*n*	Determines the number of points that subscripts and superscripts are reduced. The default is 3 points.

—s*n* Changes globally, the point size of text. *n* can be positive or negative.

CAVEATS

To embolden digits, parentheses, etc., it is necessary to quote them, as in **bold "12.3"**.
See also *CAVEATS* under *nroff*(1).

SEE ALSO

checkeq(1), eqn(1), mm(1), mmt(1), nroff(1), tbl(1), troff(1), eqnchar(7). mm(7), mv(7).

REFERENCES

Typesetting Mathematics—User Guide by B. W. Kernighan and L. L. Cherry.

NAME

netstat — show network status

SYNOPSIS

netstat [**—Aahimnrstu**] [**—a**] [*interval*] [*system*] [*core*]

DESCRIPTION

The **netstat** command symbolically displays the contents of various network–related data structures.

OPTIONS

—A
show the address of any associated protocol control blocks; used for debugging

—a show the state of all sockets; normally sockets used by server processes are not shown

—h show the state of the IMP host table [VAX on ARPANET only].

—i show the state of interfaces which have been auto–configured (interfaces statically configured into a system, but not located at boot time are not shown)

—m
show statistics recorded by the memory management routines (the network manages a "private share" of memory)

—n show network addresses as numbers (normally **netstat** interprets addresses and attempts to display them symbolically)

—r show the routing tables

—s show per–protocol statistics; with **—r**, show routing statistics

—t with **—i**, also show timer statistics

—u show unix domain socket information.

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

If an *interval* is specified, **netstat** will continuously display the information regarding packet traffic on the configured network interfaces, pausing *interval* seconds before refreshing the screen.

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 are of the form *host.port* or *network.port* if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases */etc/hosts* and */etc/networks*, respectively. If a symbolic name for an address is unknown, or if the **—n** option is specified, the address is printed in the Internet "dot format"; refer to *inet*(3n) for more

information regarding this format. Unspecified, or "wildcard", addresses and ports appear as "*".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network address (currently Internet specific) of the interface and the maximum transmission unit ("mtu") are also displayed. Additionally specifying the **—t** option will cause watchdog timer statistics to be included in the interface display.

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 to a gateway ("G"). 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.

When **netstat** is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column summarizing information for all interfaces, and a column for the interface with the most traffic since the system was last rebooted. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[INTERNAL]	An unexpected error occurred. Execution was terminated. Record the message and save the core file for analysis. Contact service personnel at your Tektronix field office.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

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

SEE ALSO

iostat(1), vmstat(1), hosts(5n), networks(5n), protocols(5n), services(5n).

NAME

newaliases — rebuild the data base for the mail aliases file

SYNOPSIS

newaliases

DESCRIPTION

Newaliases rebuilds the random access data base for the mail aliases file */usr/lib/aliases*. It is run automatically on the first attempt to send mail after */usr/lib/aliases* is changed and the change then takes effect. If the aliases file is out of date, then **sendmail** rebuilds it.

SEE ALSO

aliases(5), sendmail(8).

NAME

next — show the next mail message

SYNOPSIS

next [+*folder* **] [–header] [–noheader] [–format]**
[–noformat] [–pr] [–nopr] [–help]
[switches for **pr]**

DESCRIPTION

Next displays the next mail message in the specified (or current) folder. This command is the same as **show next**. See *show(1mh)*.

If you specify the **—pr** option, the message is displayed by **pr** and any unknown switches are passed on to **pr**.

Your .mh_profile can contain the following entries:

Path: To determine the user's MH directory
Current–Folder: To find the default current folder

If a folder is specified, it becomes the current folder, and the displayed message becomes the current message.

OPTIONS

—format
Pass the message through the default pagination program.

—noformat
Pass the message through the **cat** program.

—header
Display a header describing what message is being displayed. The header is in the following format:

(Message *folder:number*)

—noheader
Don't display the header described for the **—header** option.

—help
Display a usage message for **next**.

—pr
Use **pr** as the pagination program.

—nopr
Use the default pagination program (same as **—format**).

FILES

$HOME/.mh_profile The user profile

SEE ALSO

cat(1), comp(1mh), fmt(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), mh(1mh), mhl(1mh), next(1mh), pick(1mh), pr(1), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh), mh(5mh), mh_profile(5mh).

NAME

nice — run a command at low priority (*sh* only)

SYNOPSIS

nice [—*number*] *command [arguments]*

DESCRIPTION

Nice executes *command* with low scheduling priority. If the *number* argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 20. The default *number* is 10. The normal priority for most processes is 0.

The super–user may run commands with priority higher than normal by using a negative priority, e.g. '——10'.

Processes that have been *niced* are marked specially in *ps(1)* listings.

OPTIONS

—*number*

The priority is incremented (higher numbers mean lower priorities) by *number* up to a limit of 20. The default *number* is 10.

EXAMPLES

The following example runs the command *date(1)* at the current priority plus 12.

```
nice -12 date
```

RETURN VALUE

Nice returns the exit status of the executed command or one of the following.

[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

date(1), ps(1), renice(1), getpriority(2), setpriority(2).

NAME

nice — run a command at a different priority (**csh** built-in)

SYNOPSIS

nice [{+–}*number*] [*command* [*args...*]]

DESCRIPTION

With no *command* arguments, **nice** raises or lowers the priority of the current shell. With a *command,* that command is run in a separate shell with the priority raised or lowered. Non–superusers may only lower the priority, and thus may only use the '+' before the number. The superuser may raise the priority by using the '–' before the number.

The default number argument is '+4'.

The valid range of priorities is –20 to 20, with –20 being the highest, 20 the lowest, and 0 the normal default priority.

OPTIONS

+number

Make the new priority equal to the old priority plus the given number, with a maximum of 20.

—number

Make the new priority equal to the old priority minus the given number, with a minimum of —20.

EXAMPLES

To run the command "make myprog" at a priority that is 4 lower than the current priority, use either of the following command lines.

```
nice +4 make myprog
nice make myprog
```

RETURN VALUE

The return value is the value returned by the command executed (0 if no command), or 1 if the command was not found.

CAVEATS

The syntax for **nice** is very different from *nice(1).*

Once the current shell is niced down, its priority can only be raised by the superuser, so it is recommended that *nice(1)* be used in an alias for **nice** so that accidents are avoided.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), ps(1), pushd(1csh), rehash(1csh), renice(1), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), getpriority(2), setpriority(2).

NAME

nm — print name list

SYNOPSIS

nm [**—a**] [**—g**] [**—n**] [**—o**] [**—p**] [**—r**] [**—u**]
[*filename* ...]

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 *filename* 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), **C** (common symbol), **f** file name, or — for *sdb(1)* symbol table entries (see **—a** below). If the symbol is local (non–external) the type letter is in lower case. The output is sorted alphabetically.

Nm works with both long and short format archive files.

OPTIONS

—a Print *sdb(1)* symbol table entries. Normally, these entries are not listed.

—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.

—p Don't sort; print in symbol–table order.

—r Sort in reverse order.

—u Print only undefined symbols.

EXAMPLES

```
nm -u
```

prints only the undefined symbols in *a.out,* the default output file for the C compiler.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

[P_WARN] A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

[P_ERR] A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

ar(1), sdb(1), ar(5), a.out(5), stab(5).

NAME

nohup — run a command immune to hangup and terminate signals (*sh* only)

SYNOPSIS

nohup command [arguments]

DESCRIPTION

Nohup executes *command* immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. **Nohup** should be invoked from the shell with '&' to prevent it from responding to the next person who logs in on the same terminal. If the standard output and standard error are connected to a terminal, the standard output and standard error for the command are redirected to the file *nohup.out.*

EXAMPLES

The following example executes *nroff(1)* on the file text and sends the output to the line printer, even if the user logs off. Any messages that would be sent to the terminal are put in the file *nohup.out.*

```
nohup nroff -Tlp text&
```

Note that this command is being run in the background so the user's terminal is free while the command executes.

FILES

nohup.out	default standard output and standard error file

RETURN VALUE

Nohup returns the exit status of the command executed.

SEE ALSO

nice(1), nroff(1), renice(1), sh(1sh), getpriority(2), setpriority(2).

NAME

nohup — ignore hangup signals (**csh** built-in)

SYNOPSIS

nohup [*command* [*args...*]]

DESCRIPTION

With no arguments, **nohup** causes hangups to be ignored for the remainder of a shell script. With command arguments, the given command is run with hangups ignored. When a command is run in the background, it is already effectively **nohup**'ed.

EXAMPLES

The following command line runs the command "make myprog" and makes it immune to hangups.

```
nohup make myprog
```

DIAGNOSTICS

nohup: Can't from terminal.

The **nohup** command can not be run without arguments from an interavtive shell.

RETURN VALUE

The return value is that of the command executed (or 0 if no command), unless an error occurs, in which case the return value is 1.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), stty(1), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), signal(3c).

NAME

notify — notify of job completion (**csh** built-in)

SYNOPSIS

notify [**%***job*...]

DESCRIPTION

The **notify** command causes the shell to notify the user whenever the given jobs (or the current job if none are given) change status (see *jobs(1csh)*). Without this, status changes are printed before the next prompt. If the shell variable *notify* is set, **notify** is not needed, as all jobs are set up this way.

EXAMPLES

Assume that the job "make myprog" is job number 1, which also happens to be the current job, and that it is the only *make(1)* job running. The following shows a number of ways to turn on instant notification for this job.

```
notify
notify %1
notify %+
notify %make
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the message occurred.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), make(1), nice(1csh), nohup(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

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*, may appear in any order, but must appear before the *filenames*.

OPTIONS

—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 *CAVEATS* 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. 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*filename*
Prepend to the input *files* the non–compacted (ASCII text) macro file */usr/lib/tmac/tmac. filename*.

—c*filename*
Prepend to the input *files* the compacted macro files */usr/lib/macros/cmp. [nt] .[dt] .filename* and */usr/lib/macros/ucmp. [nt] .filename* .

—k*filename*
Compact the macros used in this invocation of **nroff**, placing the output in files [**dt**].*filename* in the current directory (see the May 1979 Addendum to the *NROFF/TROFF User's Manual* for details of compacting macro files).

—T*filename*
Prepare output for specified terminal. Known *filename*s are:

37 TELETYPE® Model 37 terminal.

ascii
basic ASCII terminal (default).

aaa
Ann Arbor Ambassador terminal.

vt100
DEC VT–100 compatible terminals.

4025
Tektronix 4025 terminal.

tn300
GE TermiNet 300 (or any terminal without half–line capability).

300s
DASI 300s terminal.

300s–12
DASI 300s terminal in 12 pitch mode.

300
DASI 300 terminal.

300–12
DASI 300 terminal in 12 pitch mode

450
DASI 450 terminal.

450–12
DASI 450 terminal in 12 pitch mode.

450–12–8
DASI 450 terminal in 12 pitch 8 lines per inch mode.

lp (generic) ASCII line printer.

lp–8
Printronix printer in 8 lines per inch mode.

lp–t
Printronix printer in 3 lines per inch mode.

lp–t–8
Printronix printer in 4 lines per inch mode.

q–8
Qume Sprint–5 printer in 12 pitch and 8 lines per inch mode.

q–10
Qume Sprint–5 printer in 10 pitch and 8 lines per inch mode.

q-lg
Qume Sprint-5 printer (Letter Gothic 12 print wheel) in 12 pitch and 6 lines per inch mode.

q-lg-8
Qume Sprint-5 printer (Letter Gothic 12 print wheel) in 12 pitch and 8 lines per inch mode.

q-lg-10
Qume Sprint-5 printer (Letter Gothic 12 print wheel) in 10 pitch and 6 lines per inch mode.

382
DTC-382.

4000a
Trendata 4000A.

832
Anderson Jacobson 832.

X EBCDIC. printer (generic)

2631
Hewlett Packard 2631 line printer.

2631-c
Hewlett Packard 2631 line printer in compressed mode.

2631-e
Hewlett Packard 2631 line printer in expanded mode.

up Standard line printer driver in 10 pitch and 6 lines per inch mode.

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

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

—u*n*
Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

FILES

/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**

RETURN VALUE

[NO_ERRS]	Command completed without error.
[nonzero]	An error of the type described occurred.

CAVEATS

When **nroff** is used with the **—o***list* option inside a pipeline (e.g., with one or more of *neqn*(1), and *tbl*(1)), it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

SEE ALSO

col(1), eqn(1), greek(1), mm(1). tbl(1), man(7), me(7), mm(7), ms(7).

NAME

od — octal, decimal, hex, ascii dump

SYNOPSIS

od [—format] [*filename*] [[+]offset[.][**b**] [label]]

DESCRIPTION

Od displays *file*, or its standard input, in one or more dump formats as selected by the first argument. If the first argument is missing, **—o** is the default.

The *offset* argument specifies the byte offset into the file where dumping is to commence. This argument is normally interpreted as octal bytes. A different radix can be specified: If "**.**" is appended to the argument, then *offset* is interpreted in decimal. If *offset* begins with "**x**" or "**0x**", it is interpreted in hexadecimal. If "**b**" ("**B**") is appended, the offset is interpreted as a block count, where a block is 1024–bytes. If the *filename* argument is omitted, an *offset* argument must be preceded by "+".

The radix of the displayed address will be the same as the radix of the *offset*, if specified; otherwise it will be octal.

Label will be interpreted as a pseudo–address for the first byte displayed. It will be shown in "()" following the file offset. It is intended to be used with core images to indicate the real memory address. The syntax for *label* is identical to that for *offset*.

By default, display lines that are identical to the last line shown are not output, but are indicated with an "*" in column 1. This behavior can be turned off with the **—v** option.

OPTIONS

—a Interpret bytes as characters and display them with their ACSII names. If the **p** character is given also, then bytes with even parity are underlined. The **P** character causes bytes with odd parity to be underlined. Otherwise the parity bit is ignored.

—b Interpret bytes as unsigned octal.

—c Interpret bytes as ASCII characters. Certain non–graphic characters appear as C escapes: null = \0, backspace = \b, formfeed = \f, newline = \n, return = \r, tab = \t; others appear as 3–digit octal numbers. Bytes with the parity bit set are displayed in octal.

—d Interpret (short) words as unsigned decimal.

—f Interpret long words as floating point.

—h Interpret (short) words as unsigned hexadecimal.

—i Interpret (short) words as signed decimal.

—l Interpret long words as signed decimal.

—o Interpret (short) words as unsigned octal.

—s[n]
Look for strings of ascii graphic characters, terminated with a null byte. *N* specifies the minimum length string to be recognized. By default, the minimum length is 3 characters.

—v Show all data. By default, display lines that are identical to the last line shown are not output, but are indicated with an "*" in column 1.

—w[n]
Specifies the number of input bytes to be interpreted and displayed on each output line. If **w** is not specified, 16 bytes are read for each display line. If *n* is not specified, it defaults to 32.

—x Interpret (short) words as hexadecimal.

An upper case format character implies the long or double precision form of the object.

EXAMPLES

The following invocation of od produces an octal dump in 132 column format, which works nicely if printed on wide printer paper.

```
od -ow frmt
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

A hexadecimal offset can't be a block count.

It is an historical botch to require specification of object, radix, and sign representation in a single character argument.

SEE ALSO

adb(1), sdb(1).

NAME

onintr — interrupt handling (**csh** built-in)

SYNOPSIS

onintr [—]
onintr [*label*]

DESCRIPTION

The command **onintr** is used in shell scripts to handle interrupts. If no arguments are given, the default interrupt handling, which is usually to abort the program, is restored. With the argument —, interrupts are ignored. With a *label* argument, interrupts cause the command **goto** *label* to be executed. Interrupts may either be received by the shell or by a child process to be handled by the **onintr** control.

If a shell is running detached and interrupts are being ignored, **onintr** commands are ignored.

EXAMPLES

The following shell script builds a file from all files named on the command line, and moves the file to the file named by the environment variable OUTPUTFILE. The resulting file contains no lines beginning with the character '#'. The **onintr** command is used to abort the script if an interrupt is received during the file building, and is used to ignore interrupts during the file move.

```
#!/bin/csh -f
#
# First, make sure that the variable OUTPUTFILE is set.
#
if ($?OUTPUTFILE == 0) then
        echo "$0 : OUTPUTFILE variable not set."
        exit 1
endif

onintr abort

cp /dev/null /tmp/output.$$
foreach name ($argv)
        grep -v '^#' "$name" >> /tmp/output.$$
end

onintr -
mv /tmp/output.$$ "$OUTPUTFILE"
exit
#
# Interrupt handling.
#

abort:
```

```
echo "$0 : Aborting. $OUTPUTFILE not updated."
exit 1
```

DIAGNOSTICS

onintr: Can't from terminal.

The **onintr** command can only be used in shell scripts. This message occurs whenever **onintr** is invoked from a terminal.

RETURN VALUE

[NO_ERRS] Command completed without error.

[1] The command was invoked from a terminal.

CAVEATS

In *csh(1csh),* only interrupts and hangups (see *nohup(1csh))* can be ignored, and only interrupts can be caught. If more control over signals is required, *sh(1sh)* should be used.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), trap(1sh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), signal(3c).

NAME

more, page — file perusal filter for crt viewing

SYNOPSIS

more [**—c**] [**—d**] [**—f**] [**—l**] [**—p**] [**—s**] [**—u**] [*—n*] [*+linenumber*] [*+/pattern*] [*filename...*]

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a soft-copy 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. Other possibilities are enumerated later.

If the program is invoked as **page,** or with the **—p** option, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and k — 1 rather than k — 2 lines are printed in each screenful, where k is the number of lines the terminal can display.

More looks in the termcap entry (see TERMCAP in the VARIABLES section) to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre—set any flags desired. For example, if you prefer to view files using the *—c* mode of operation, the *sh* command sequence *MORE='—c' ; export MORE* would cause all invocations of **more** , including invocations by programs such as *man* and *msgs* , to use this mode. The *MORE* environment variable is only checked for flags, so the preceding — is not required. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.profile* file.

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

Other sequences which may be typed when **more** pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i⟨space⟩
: display *i* more lines, (or another screenful if no argument is given)

i⟨cr⟩
: display *i* more lines, (or one line if no argument is given). The value of *i* becomes the new window size.

^D
: display 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d
: same as ^D (control—D)

*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

q or Q Exit from **more.**

= Display the current line number.

v Start up the editor *vi* at the current line.

h or ? Help command; give a description of all the **more** commands.

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.

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

' (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 characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n skip to the *i*—th next file given in the command line (skips to last file if i 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.

:f display the current file name and line number.

:q or :Q exit from **more** (same as q or Q).

. (dot) repeat the previous 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 (see *stty(1)*) 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).

A sample usage of **more** in previewing *nroff* output would be

```
nroff -ms +2 doc.n ¦ more -s
```

OPTIONS

—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 **More** will prompt the user with the message "Hit space to continue, Q or q to quit" at the end of each screenful. This is useful if **more** is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

—f This 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 Do not treat ^L (form feed) specially. If this option is not given, **more** will pause after any line that contains a ^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.

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

—p **More** will erase the entire screen before displaying the next page. This is the same as executing **page.**

—s Squeeze multiple empty lines from the output, producing only one empty line (lines with whitespace are not empty). Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

—u Normally, **more** will handle underlining and bold characters 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, bold or stand—out mode for underlined information in the source file. The *—u* option suppresses this processing.

+linenumber
Start up at *linenumber.*

+/pattern
Start up three lines before the line containing the regular expression *pattern.*

EXAMPLES

The following example will display the file text.c starting at line 65.

```
more +65 text.c
```

FILES

/etc/termcap	Default terminal data base.
/usr/lib/more.help	Help file.

VARIABLES

MORE	The options to be used when invoking more.
SHELL	The user's login shell. Used for shell escapes.
TERM	The type of terminal being used.
TERMCAP	The name of the file containing the terminal capability entry, or the entry itself.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

If no *ho* string exists in the termcap entry, cursor motion is used to home the cursor when the **-c** or **-p** options are used. In these cases, the *ti* string is printed before text is displayed, and the *te* string is printed before **more** exits. The manual page for *termcap(5t)* describes *ti* and *te* as the

strings required to enter/exit programs that use cursor motion. This means that *te* should not clear the screen! If it does, your screen will be cleared when you exit **more.**

Lines longer than 1024 characters are separated into 1024 character lines separated by newlines. The 1024 character limit applies to input characters, so a line that contains a large number of backspaces may be separated even if it prints as less than 1024 characters.

When **more** is run on files that are being written or appended, the percentage of the file viewed may be listed as more than 100%.

Numbers in the options are scanned sequentially and do not get reset. For example, the commands

```
more -1f1
more -11f
```

are equivalent.

When a line contains a pagefeed (^L), the entire line is displayed at the end of the page, instead of being split up.

SEE ALSO

man(1man), nroff(1), page(1), sh(1sh), stty(1), ul(1), termcap(5t), environ(7).

NAME

pair — pair element groups

SYNOPSIS

pair [**—c***n*] [**—F***vector*] [**—x***n*] [*vector ...*]

DESCRIPTION

Output is a vector with elements taken alternately from a *base* vector and from a *vector*. The *base* vector is specified either with the F option, or else it comes from the standard input. *Vector(s)* are specified either on the command line or else one may come from the standard input. If both the *base* and *vector* come from the standard input, *base* precedes *vector*.

OPTIONS

—c*n*
n is the number of output elements per line.

—F*vector*
vector is the *base*.

—x*n*
n is the number of elements taken from the *base* for each one element taken from *vector*.

EXAMPLES

The following example outputs a vector with three elements from A, then one from B, then three from A, one from B, and so on.

```
pair -x3,FA B
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

passwd — change login password

SYNOPSIS

passwd [*userid*]

DESCRIPTION

This command changes (or installs) a password associated with the user *userid* (your own user name by default).

The program prompts for the old password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. At least one non–numeric character is required. These rules are relaxed if you are insistent enough.

Only the owner of the name or the super–user may change a password; the owner must prove he knows the old password.

EXAMPLES

To change your own password, execute the command **passwd** with no arguments. You will be prompted for all required information.

```
passwd
```

FILES

/etc/passwd	System user account information file

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

login(1), crypt(3c), passwd(5).

NAME

paste — merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste [—dlist] *file1 file2 ...*
paste —s [—dlist**]** *file1 file2 ...*

DESCRIPTION

In the first form, **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). It is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the second 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.

OPTIONS

—d Without this option, the new–line characters of each but the last file (or last line in case of the **—s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).

list One or more characters immediately following **—d** replace the default *tab* as the line concatenation character. The list is used circularly, i. e. when exhausted, it is reused. In parallel merging (i. e. no **—s** option), the lines from the last file are always terminated with a new–line character, not from the *list*. The list may contain the special escape sequences: **\n** (new–line), **\t** (tab), \\ (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g. to get one backslash, use "" *—d*"\\\\").

—s Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **—d** option. Regardless of the *list*, the very last character of the file is forced to be a new–line.

— May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

```
ls | paste -d" " -
```

lists directory in one column

```
ls | paste - - - -
```

lists directory in four columns

```
paste -s -d"\ t\ n" file
```

combines pairs of lines into single lines

```
pr -t -m file
```

works like **paste** but creates extra blanks, tabs, and newlines for a nice page layout.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Line length is limited to 1023 characters. An error will be produced if a line of greater lingth is encountered.

With the exception of usage with the —**s** option, no more than 17 input files may be specified.

SEE ALSO

awk(1), comm(1), cut(1), egrep(1), fgrep(1), grep(1), join(1), look(1), sort(1), uniq(1).

NAME

pathof — print the path of a command

SYNOPSIS

pathof [**-a**] [**-s**] *command* ...

DESCRIPTION

For each *command* name given, **pathof** searches all directories named in the environment variable $PATH for an executable file by that name. If such a file is found, its pathname is written to standard out. Otherwise an error message is written to standard error. If the **—s** option is used, no messages are written; only an exit status is returned.

OPTIONS

—a All paths for that command are found rather than just the one that would currently be executed.

—s No messages are written; only an exit status is returned.

EXAMPLES

Pathof is useful in finding what version of a program you are executing.

```
pathof a.out
```

Finding whether a program exists by a particular name:

```
if pathof -s a.out; then...
```

Editing shell scripts:

```
vi `pathof nohup`
```

VARIABLES

PATH	The user's execution search path.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	At least one of the commands specified was not found.
[USAGE]	Incorrect command line syntax. Execution terminated.

SEE ALSO

csh(1csh), sh(1sh), type(1sh), which(1csh), which(1sh).

NAME

pc — Pascal compiler

SYNOPSIS

pc [option] [**—i** *filename...*] *filename...*

DESCRIPTION

Pc is a Pascal compiler. If given an argument file ending with **.p**, it will compile the file and load it into an executable file called, by default, *a.out*.

A program may be separated into more than one **.p** file. **Pc** will compile a number of argument **.p** files into object files (with the extension **.o** in place of **.p**). Object files may then be loaded into an executable *a.out* file. Exactly one object file must supply a **program** statement to successfully create an executable a.out file. The rest of the files must consist only of declarations which logically nest within the program. References to objects shared between separately compiled files are allowed if the objects are declared in **include**d header files, whose names must end with **.h**. Header files may only be included at the outermost level, and thus declare only globally available objects. To allow **function**s and **procedure**s to be declared, an **external** directive has been added, whose use is similar to the **forward** directive but restricted to appear only in **.h** files. **Function** and **procedure** bodies may not appear in **.h** files. A binding phase of the compiler checks that declarations are used consistently, to enforce the type checking rules of Pascal.

Object files created by other language processors may be loaded together with object files created by **pc**. The **function**s and **procedure**s they define must have been declared in **.h** files included by all the **.p** files which call those routines. Calling conventions are as in C, with **var** parameters passed by address.

See the *Berkeley Pascal User's Manual* for details.

OPTIONS

The following options have the same meaning as in *cc*(1) and *f77*(1). See *ld*(1) for load-time options.

—c Suppress loading and produce '.o' file(s) from source file(s).

—go
Have the compiler produce additional symbol table information for *sdb*(1).

—o output
Name the final output file *output* instead of *a.out*.

—p Prepare object files for profiling, see *prof*(1).

—pg
Prepare object files for profiling, see *gprof*(1).

—w
Suppress warning messages.

—O Invoke an object-code improver.

—S Compile the named program, and leave the assembler–language output on the corresponding file suffixed '.s'. (No '.o' is created.).

The following options are peculiar to **pc**.

—b Block buffer the file *output.*

—i Produce a listing for the specified procedures, functions and **include** files.

—l Make a program listing during translation.

—s Accept standard Pascal only; non–standard constructs cause warning diagnostics.

—sc
Make the compiler case insensitive, but allow all other extensions.

—C
Compile code to perform runtime checks, verify **assert** statements, and initialize all variables to zero

Other arguments are taken to be loader option arguments, perhaps libraries of **pc** compatible routines. Certain flags can also be controlled in comments within the program as described in the *Berkeley Pascal User's Manual.*

FILES

file.p	pascal source files
/usr/lib/pc0	compiler
/lib/f1	code generator
/usr/lib/pc2	runtime integrator (inline expander)
/lib/c2	peephole optimizer
/usr/lib/pc3	separate compilation consistency checker
/usr/lib/pc2.0strings	text of the error messages
/usr/lib/how_pc	basic usage explanation
/usr/lib/libpc.a	intrinsic functions and I/O library
/usr/lib/libm.a	math library
/lib/libc.a	standard library, see *intro*(3)

DIAGNOSTICS

For a basic explanation do

pc

In the diagnostic output of the translator, lines containing syntax errors are listed with a flag indicating the point of error. Diagnostic messages indicate the action which the recovery mechanism took in order to be able to continue parsing. Some diagnostics indicate only that the input is 'malformed.' This occurs if the recovery can find no simple correction to make the input syntactically valid.

Semantic error diagnostics indicate a line in the source text near the point of error. Some errors evoke more than one diagnostic to help pinpoint the error; the follow–up messages begin with an ellipsis '. . . '.

The first character of each error message indicates its class:

E Fatal error; no code will be generated.

e Non–fatal error.

w Warning — a potential problem.

s Non–standard Pascal construct warning.

Internal errors cause messages containing the word SNARK.

CAVEATS

The keyword **packed** is recognized but has no effect.

The binder is not as strict as described here, with regard to the rules about external declarations only in '.h' files and including '.h' files only at the outermost level. It will be made to perform these checks in its next incarnation, so users are warned not to be sloppy.

Because the **—s** option is usurped by the compiler, it is not possible to pass the strip option to the loader. Thus programs which are to be stripped, must be run through *strip*(1) after they are compiled.

SEE ALSO

prof(1) sdb(1).

NAME

pd — **plot** dump

SYNOPSIS

pd [*plot file* ...]

DESCRIPTION

Pd prints a human readable listing of **plot** or **tplot** format graphical commands. If no *file* is given the standard input is assumed.

Plot is a graphics utility and library that is part of 4.2 BSD UNIX. In System V UNIX, it is known as **tplot**. Neither is included with the current release of UTek. **Pd** is included in the UTek Graphics Tools for compatibility with these other versions of UNIX.

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

pick — select mail messages by content

SYNOPSIS

pick [msgs] [-help] [-src +folder]
[-cc pattern] [-date pattern] [-from pattern]
[--*component* pattern] [-search pattern] [-subject pattern]
[-to pattern]

[[-file [-link] [-nolink] [-preserve] [-nopreserve] +folder ...]]
[[-keep [-stay] [-nostay] [+folder ...]]
[-nofile] [-nokeep]

[-scan] [-noscan] [-show] [-noshow] [-stay] [-nostay]

DESCRIPTION

Pick searches mail messages within a folder for the specified contents, then performs operations on the selected messages.

A modified *grep*(1) is used to perform the searching, so the full regular expression (see *ed*(1)) facility is available within 'pattern'. With **—search**, pattern is used directly, and with the others, the grep pattern constructed is:

"component:.*pattern"

This means that the pattern specified for a **—search** is found everywhere in the message, including the header and the body, while the other search requests are limited to the single specified component. The expression **——component pattern** is a shorthand for specifying '—search "component:.*pattern" '; it is used to pick a component not in the set [cc date from subject to]. An example is "pick ——reply—to pooh —show".

Searching is done on a per-line basis. Within the header of the message, each component is treated as one long line, but in the body, each line is separate. Lower-case letters in the search pattern match either lower or upper case in the message, while upper case match only upper case.

Once the search has been performed, the selected messages are scanned (see *scan*(1mh)) if the **—scan** switch is given, and then they are shown (see *show*(1mh)) if the **—show** switch is given. After these two operations, the file operations (if requested) are performed.

The **—file** switch operates exactly like the *refile* command, with the same meaning for the **—preserve** and **—link** switches.

The **—keep** switch is similar to **—file**, but it produces a folder that is a subfolder of the folder being searched and defines it as the current folder (unless the **—stay** flag is used). This subfolder contains the messages which matched the search criteria. All of the MH commands may be

used with the sub-folder as the current folder. This gives you considerable power in dealing with subsets of messages in a folder.

The messages in a folder produced by **—keep** always have the same numbers as they have in the source folder (i.e., the **—preserve** switch is automatic). This way, the message numbers are consistent with the folder from which the messages were selected. Messages are not removed from the source folder (i.e., the **—link** switch is assumed). If +folder is not specified, the standard name *select* is used. (This is the meaning of *select* when it appears in the output of the *folder* command.) If +folder arguments are given to **—keep**, they are used rather than *select* for the names of the subfolders. This allows for several subfolders to be maintained concurrently.

Your .mh_profile can contain the following entries:

Path: To determine the user's MH directory
Folder—Protect: For protection on new folders
Current—Folder: To find the default current folder

pick has the following defaults:

—src +folder defaults to current
msgs defaults to all
—keep +select is the default if no **—scan,—show**, or**—file** is specified

If a **—src** +folder is specified, *folder* becomes the current folder, unless a **—keep** with 0 or 1 folder arguments makes the selection-list subfolder the current folder. Each selection-list folder has its current message set to the first of the messages linked into it unless the selection list already existed, in which case the current message isn't changed.

OPTIONS

—cc *pattern*
Search for the specified name in the "Cc:" line of the message header.

—src +folder
Search *folder* **for the messages.**

—help
Print help information.

—scan
Display a scan listing of the selected messages.

—noscan
Don't display a scan listing of the selected messages.

—date *pattern*
Search the Date: fields of mail messages for the specified date.

—show
Display the selected messages.

—noshow
Don't display the selected messages.

—nofile
Opposite of **—file**.

—nokeep
Opposite of **—keep**.

—from pattern
Search the From: fields of mail messages for the specified sender.

—search *pattern*
Search the entire mail message for *pattern*. **The other options that specify a search pattern restrict their search to the specified field of the mail message.**

—subject *pattern*
Search the Subject: fields of mail messages for the specified subject.

—to *pattern*
Search the To: fields of mail messages for the specified recipient.

—file
Operates exactly like the **refile** command, with the same meaning for the **—preserve** and **—link** switches. See *refile(1mh)*.

—preserve
When moving or linking a mail message to another mail folder, give the new message the same number as it had in the old folder. This option works with the **—file** option.

—link
Link (don't move) the selected mail messages into another folder. This option works with the **—file** option.

—nopreserve
When moving or linking a mail message to another mail folder, give the new message a number one higher than the highest numbered message currently in the new folder. This option works with the **—file** option.

—nolink
Move (don't link) the selected mail messages into another folder. This option works with the **—file** option.

——component pattern
Search the field corresponding to *component* for *pattern*.

—keep
Similar to **—file**, but it produces a folder that is a subfolder of the folder being searched and defines the new folder as the current folder (unless you specify **—stay**).

—stay
Used with the **—keep** flag to prevent the newly created folder from becoming the current folder.

—nostay
Used with the **—keep** flag to make the newly created folder the current folder. This is the default for **—keep**.

EXAMPLES

The first example displays a scan listing of all mail messages from *jones*.

```
pick -from jones -scan
```

This example displays the number of messages addressed to *holloway*.

```
pick -to holloway
```

This example produces a scan listing of all the mail messages whose subject contains *ned*. The matching messages are moved into the subfolder *select*.

```
pick -subject ned -scan -keep
```

This example shows the power of the MH package. In item 1, the current folder is set to inbox. In 3, all of the messages from dcrocker are found in inbox and linked into the folder 'inbox/select'. (Since no action switch is specified, **—keep** is assumed.) Items 6 and 7 show that this subfolder is now the current folder. Items 8 through 14 are a *scan* of the selected messages (note that they are all from dcrocker and are all in upper and lower case). Item 15 lists all of the messages to the high-speed printer. Item 16 directs *folder* to set the current folder to the parent of the selection-list folder, which is now current. Item 17 shows that this has been done. Item 18 resets the current folder to the selection list, and 20 removes the selection-list folder and resets the current folder to the parent folder, as shown in 21 and 22.

```
 1 $ folder +inbox
 2 inbox+ has 16 messages (  3- 22); cur= 3.
 3 $ pick -from dcrocker
 4 6 hits.
 5 [+inbox/select now current]
 6 $ folder
 7 inbox/select+ has  6 messages (  3- 16); cur= 3.
 8 $ scan
 9  3+ 6/20 Dcrocker Re: ned file update issue...
10  6  6/23 Dcrocker removal of files from /tm...
11  8  6/27 Dcrocker Problems with the new ned...
12 13  6/28 dcrocker newest nned ⟨⟨I would ap...
13 15  7/ 5 Dcrocker nned ⟨⟨Last week I asked...
14 16  7/ 5 dcrocker message id format ⟨⟨I re...
15 $ show all ¦ print
16 $ folder -up
17 inbox+ has 16 messages (  3- 22); cur= 3; (select).
18 $ folder -down
19 inbox/select+ has 6 messages (  - 16); cur= 3.
20 $ rmf
```

```
21 [+inbox now current]
22 $ folder
23 inbox+ has 16 messages (  3- 22); cur= 3.
```

FILES

$HOME/.mh_profile The user profile

SEE ALSO

comp(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh).

NAME

pie — build a pie chart

SYNOPSIS

pie [**—b**] [**—p**] [**—pp***n*] [**—pn***n*] [**—v**] [**—o**] [**—r***n*] [**—x***n*] [*vector ...*]

DESCRIPTION

Output is a GPS that describes a pie chart. If no *file* is specified, the standard input is assumed. Input is lines of the form:

[⟨*control*⟩] *value* [*label*]

Square brackets represent optional fields. Angle brackets (⟨⟩) are literal.

Each line of the input specifies a single slice of the pie. The pie is drawn with the *value* of each slice printed inside and the *label* printed outside. The *control* field specifies the appearance of the slice. Legal values for the control field are:

i The slice is not drawn, though a space is left for it.

e The slice is "exploded", or moved away from the pie.

f The slice is filled. The angle of fill lines depends on the color of the slice.

c*color*
The slice is drawn in *color* rather than the default black. Legal values for *color* are **b** for black, **r** for red, **g** for green, and **u** for blue.

For example, the input line

```
⟨ecg⟩ 15 profit
```

specifies an exploded slice in the color green, with a value of 15 and a label of **profit**. The size of the slice depends on the values of the other slices of the pie.

OPTIONS

—b Draw pie chart in bold weight lines, otherwise use medium.

—p Output *value* as a percentage of the total pie.

—pp*n*
Only draw *n* percent of a pie.

—pn*n*
Output *value* as a percentage, but total of percentages equals *n* rather than 100. **–pn100** is equivalent to **–p**.

—v Do not output values.

—o Output values around the outside of the pie.

—r*n*
Put the pie chart in region *n* of the GPS universe, where *n* is between 1 and 25 inclusive.

—x*n*
Position the pie chart in the GPS universe with x–origin at *n*.

—y*n*
Position the pie chart in the GPS universe with y–origin at *n*.

EXAMPLES

The following example will draw the pie chart specified by file A in 80% of a circle and will output the *values* as percentages that total 80.

```
pie -pp80,pn80 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

plot — plot an x–y graph

SYNOPSIS

plot [**—a**] [**—b**] [**—c***char(s)*] [**—d**] [**—F***vector*] [**—g**] [**—m**] [**—r***n*] [**—x***n*] [**—xa**] [**—xi***n*] [**—xl***n*] [**—xh***n*] [**—xn***n*] [**—xt**] [**—y***n*] [**—ya**] [**—yi***n*] [**—yl***n*] [**—yh***n*] [**—yn***n*] [**—yt**] [*vector* ...]

DESCRIPTION

Output is a GPS that describes an x–y graph. Input is one or more *vector(s)*. Y–axis values come from *vector(s)*, x–axis values from the **–F** option. Axis scales are determined from the first *vector(s)* plotted. If no *vector* is given, the standard input is assumed.

OPTIONS

—a Suppress axes.

—b Plot graph with bold weight lines, otherwise use medium.

—c*string*
Use the characters of *string* for plotting characters, implies option **–m**. The first character of *string* is used to mark the first graph, the second is used to mark the second graph, etc.

—d Do not connect plotted points, implies option **–m**.

—F*vector*
Use *vector* for x–values, otherwise the positive integers are used.

—g Suppress background grid.

—m
Mark the plotted points.

—r*n*
Put the graph in GPS region *n*, where *n* is between 1 and 25 inclusive.

—x*n*
Position the graph in the GPS universe with x–origin at *n*.

—xa
Omit x–axis labels.

—xi*n*
n is the x–axis tick increment.

—xl*n*
n is the x–axis low tick value.

—xh*n*
n is the x–axis high tick value.

—xn*n*
n is the approximate number of ticks on the x–axis.

—xt
Omit x–axis title.

—y*n*
Position the graph in the GPS universe with y–origin at *n*.

—ya
Omit y–axis labels.

—yi*n*
n is the y–axis tick increment.

—yl*n*
n is the y–axis low tick value.

—yh*n*
n is the y–axis high tick value.

—yn*n*
n is the approximate number of ticks on the y–axis.

—yt
Omit y–axis title.

EXAMPLES

The following example plots *vector* A against the positive integers.

```
plot A
```

The following example plots *vector* B against *vector* A. Y–axis ticks begin at 0, no x–axis labels are printed, and the plot is placed in region 5 of the GPS universe.

```
plot -r5,yl0,xa,FA B
```

The following example plots *vectors* A and B against the positive integers, with y–axis ticks going from the lowest value in A and B to the highest value in A and B (see *hilo(1g)*).

```
plot -`hilo -oy A B` A B
```

The following example plots *vector* C against A and *vectors* D and E against B. Y–axis scale is determined from C, x–axis from A.

```
plot -FA,FB C D E
```

CAVEATS

This **plot** is not the same as the **plot** package that is part of 4.2 BSD UNIX (which is called **tplot** in System V UNIX).

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

point — empirical cumulative density function point

SYNOPSIS

point [—**f***n*] [—**p***n*] [—**n***n*] [—**s**] [*vector ...*]

DESCRIPTION

Output is a linearly interpolated value from the empirical cumulative density function (e.c.d.f) for the input *vector*. By default, **point** returns the median (50% point). If no *vector* is given, the standard input is assumed.

OPTIONS

—**f***n*
Return the (1/*n*)*100 percent point from the empirical cumulative density function.

—**p***n*
Return the *n**100 percent point.

—**n***n*
Return the *n*th element.

—**s** The input is assumed to be sorted.

EXAMPLES

The following example outputs the 25% point from the empirical cumulative density function. *Vector* A is assumed to be sorted.

```
point -s,p.25 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

cd, chdir, pushd, popd, dirs — directory change commands (**csh** *built-in)*

SYNOPSIS

cd [*dirname*]
or
cd *+n*

chdir [*dirname*]
or
chdir *+n*

pushd [*dirname*]
or
pushd *+n*

popd [*+n*]

dirs

DESCRIPTION

Csh maintains the name current directory and a directory stack which can be used to keep up with all of the places the user has been and may wish to go back to. With this, the user does not have to remember where to go back to when sidetracked.

The command **cd,** and its synonym **chdir,** change the current working directory of the shell. If no argument is given, the directory is changed to the user's home directory. If *dirname* begins with '/', './', or '../', an attempt is made to change to that directory. Otherwise, the directory *dirname* is searched for in the current directory and in each element of the variable *cdpath.* If this fails, and there is a variable named the same as the value of *dirname* whose value begins with a '/', that directory is used (see EXAMPLES). When the directory is changed, the top element of the directory stack is replaced by the new directory. When the *cdpath* or directory name variable is used to change the directory, the path of the new current directory is printed. With the argument *+n,* where *n* is a number (beginning at 0), the *n*'th element of the directory stack is moved from that position to the top of the stack. The contents of the directory stack is always printed upon completion.

The command **pushd** is used to add directory names to the directory stack, and to edit the stack, as well as change the current directory. With no arguments, **pushd** exchanges the top two elements of the directory stack. With a directory name argument, the directory is changed, and the name of the new current directory is pushed on to the stack. With the argument *+n,* where *n* is a number (beginning at 0), the *n*'th element of the directory stack is moved from that position to the top of the stack. The contents of the directory stack is always printed upon completion.

The command **popd** is used to remove elements from the directory stack. With no arguments, the top element of the stack is removed, resulting in changing the current working directory. With the argument *+n*, where *n* is a number (numbers begin at 0, but '+0' is not valid), the *n*'th element of the directory stack is removed. In the latter use, the current directory is unchanged. The contents of the directory stack is always printed upon completion.

The command **dirs** prints the contents of the directory stack in order. The top element (which is the current directory) is printed first.

The variable *cdpath* contains a list of directories to search if the directory name given is not a subdirectory of the current directory. This variable is maintained along with the environment variable CDPATH, which is used by the *sh(1sh)* **cd** command, but these variables are not the same. In **csh,** the current directory is implicitly the first element in *cdpath,* whereas in **sh,** the current directory must be given explicitly. In order to cope with this difference, the value of CDPATH is imported upon startup of a new shell. CDPATH is changed when *cdpath* is changed (using *set(1csh)),* but *cdpath* is not changed when CDPATH is changed (using *setenv(1csh)).*

The variable *cwd* is set by **csh** whenever the current directory is changed. Due to symbolic links and the ditributed file system, this variable may not always contain correct or desirable data. If the value of *cwd* is required to be correct, the variable *hardpaths* may be set, which causes all changes of directory to get the current directory path by calling *getwd(3c),* which will give the correct path. This makes directory changes somewhat slower. An alternate method is to use the command *pwd(1)* to get the correct current directory path when needed.

EXAMPLES

This example shows a use of the directory name variable.

```
set default=/
set cdpath=( ~ ~/* )
cd default
```

In this case, if there is no directory named 'default' in the current directory, the user's home directory, and any subdirectories of the user's home directory, the current directory will become '/'.

The following example shows some of the features of manipulation of the directory stack. Here, the character '%' represents the **csh** prompt.

```
% dirs
~

% pushd /bin
/bin ~
```

```
% pushd /etc
/etc /bin ~
% cd /usr
% dirs
/usr /bin ~
% pushd +2
~ /usr /bin
% popd +2
~ /usr
% popd
/usr
```

VARIABLES

CDPATH	The directory change search path.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the error message occurred.

CAVEATS

Shell scripts should never change to a subdirectory and attempt to go back by executing **cd .. ** , since the directory changed to may be a symbolic link to another directory whose parent directory is different from where the last **cd** was executed. At the very least, the *cwd* variable's value should be saved and used to go back.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1sh), chdir(1sh), continue(1csh), csh(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), chdir(2).

NAME

power — power function

SYNOPSIS

power [—**c***n*] [—**p***n*] [*vector ...*]

DESCRIPTION

Output is a vector with each element being a power of the corresponding element from the input *vector(s)*. If no *vector* is given, the standard input is assumed.

OPTIONS

—**c***n*

n is the number of output elements per line.

—**p***n*

Input elements are raised to the *n*th power. If not given, 2 is used.

EXAMPLES

The following example outputs the 3.5th power of each element of A, three per line.

```
power -p3.5,c3 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

pr — print files

SYNOPSIS

pr [*+num*] [*—num*] [**—a**] [**—d**] [**—e** [*(char,num)*]] [**—f**] [**—h** *hdr*] [**—i** [*(char,num)*]] [**—l** *length*] [**—m**] [**—n** [*(char,num)*]] [**—o** *offset*] [**—p**] [**—r**] [**—s** [*char*]] [**—t**] [**—w** *width*] [**—F**] [**—P**] [*file ...*]

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. The date and time is that of the last modification of the file, except when the standard input is read, in which case the current date and time is used.

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.

OPTIONS

+num
Begin printing with page *num* (default is 1).

—num
Produce *num*–column output (default is 1). The options **—e** and **—i** are assumed for multi–column output.

—a Print multi–column output across the page.

—d Double–space the output.

—e [*(char,num)*]
Expand *input* tabs to character positions *num* + 1, 2**num* + 1, 3**num* + 1, etc. If *num* 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 *char* (any non–digit character) is given, it is treated as the input tab character (default for *char* is the tab character).

—f Use form–feed character for new pages (default is to use a sequence of line–feeds).

—h *hdr*
Use *hdr* as the name to be printed in the header instead of the file name.

—i [*(char,num)*]
In *output*, replace white space wherever possible by inserting tabs to character positions *num* + 1, 2**num* + 1, 3**num* + 1, etc. If *num* is 0 or is omitted, default tab settings at every eighth position are assumed.

If *char* (any non-digit character) is given, it is treated as the output tab character (default for *char* is the tab character).

—l *length*
Set the length of a page to *length* lines (default is 66).

—m
Merge and print all files simultaneously, one per column (overrides the *—k*, and **—a** options).

—n [*(char,num)*]
Provide *num*-digit line numbering (default for *num* is 5). The number occupies the first *num* + 1 character positions of each column of normal output or each line of **—m** output. If *char* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *char* is a tab).

—o *offset*
Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the width and offset. **—p** Pause before beginning each page (including the first page) if the output is directed to a terminal (**pr** rings the bell at the terminal and waits for a carriage return).

—r Print no diagnostic reports on failure to open files or for empty files.

—s [*char*]
Separate columns by the single character *char* instead of by the appropriate number of spaces (default for *char* is a tab).

—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.

—w *width*
Set the width of a line to *width* character positions for equal-width multi-column output (default is 72). The **—w** option has no effect on single-column output.

—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.

—P
Pause after printing each page if the output is directed to a terminal (**pr** rings the bell at the terminal and waits for a carriage return).

EXAMPLES

Print *file1* and *file2* as a double-spaced, three-column listing headed by *file list:*

```
pr -3dh "file list" file1 file2
```

Write *file1* on *file2*, expanding tabs to columns 10, 19, 28, 37, ... :

```
pr -e9 -t <file1 >file2
```

FILES

*/dev/tty** to suspend messages

RETURN VALUE

[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NO_ERRS]	Command completed without error.

CAVEATS

There are no diagnostics when **pr** is printing on a terminal.

Inter–terminal messages via *write(1)* are forbidden during execution of **pr.**

SEE ALSO

cat(1), lpr(1mdqs),

NAME

prev — show the previous mail message

SYNOPSIS

prev [*+folder* **] [-header] [-noheader] [-format]**
[-noformat] [-pr] [-nopr] [-help]
[switches for **pr**]

DESCRIPTION

Prev displays the previous mail message in the specified (or current) mail folder. This is the same as **show prev**.

If you specify the **—pr** option, the message is displayed by **pr** and any unknown switched are passed on to **pr**.

Your *.mh_profile* can contain the following entries:

Path: To determine the user's MH directory
Current-Folder: To find the default current folder

If a folder is specified it becomes the current folder. The message that is shown (i.e., the previous message in sequence) becomes the current message in the folder.

OPTIONS

—format
Pass the message through the default pagination program.

—noformat
Pass the message through the **cat** program.

—header
Display a header describing which message is being displayed. The header is in the following format:

(Message *folder:number*)

—noheader
Don't display the header described for the **—header** option.

—help
Display a usage message for **prev**.

—pr
Use **pr** as the pagination program.

—nopr
Use the default pagination program (same as **—format**).

FILES

$HOME/.mh_profile The user profile

SEE ALSO

cat(1), comp(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), next(1mh), pick(1mh), pr(1), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh).

NAME

prime — generate prime numbers

SYNOPSIS

prime [—**c***n*] [—**h***n*] [—**l***n*] [—**n***n*]

DESCRIPTION

Output is a vector of *number* elements determined by the parameters *low* and *high*. The parameters are set by command options.

OPTIONS

—**c***n*
n elements per output line.

—**h***n*
high: = *n*.

—**l***n*
low: = *n*. If not given, *low*: = 2.

—**n***n*
number: = *n*. If not given, *number*: = 10.

EXAMPLES

The following example generates all prime numbers between 200 and 300.

```
prime -l200,h300
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

print — pr to the line printer

SYNOPSIS

print file ...

DESCRIPTION

Print is a shell script which paginates each file and pipes to the print queue submission program.

SEE ALSO

lpr(1mdqs), pr(1).

NAME

printenv — print out the environment

SYNOPSIS

printenv [name]

DESCRIPTION

Printenv prints out the values of the variables in the environment. If a *name* is specified and has a value, only its value is printed. Only one name may be specified. If no name is specified, all variables are printed, one per line, in the format

name = value

EXAMPLES

The following example prints the name of the user's home directory.

```
printenv HOME
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	Specified variable is not defined.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[INTERNAL]	An unexpected error occurred. Execution was terminated. Record the message and save the core file for analysis. Contact service personnel at your Tektronix field office.

SEE ALSO

set(1sh), setenv(1csh), sh(1sh), environ(7).

NAME

prod — product

SYNOPSIS

prod [*vector* ...]

DESCRIPTION

Output is the product of all the elements in the input *vector(s)*. If no *vector* is given, the standard input is assumed.

EXAMPLES

The following example outputs the product of the elements in the *vector* A.

```
prod A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

prof — display profile data

SYNOPSIS

prof [**—a**] [**—l**] [**—n**] [**—s**] [**—z**]
[**—v** [*—low* [*—high*]]] [a.out [mon.out ...]]

DESCRIPTION

Prof interprets the file produced by the *monitor* subroutine. Under default modes, the symbol table in the named object file *(a.out* default) is read and correlated with the profile file *(mon.out* default). 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 more than one profile file is specified, the output represents the sum of the profiles.

In order for the number of calls to a routine to be tallied, the **—p** option of *cc, f77* or *pc* must have been given when the file containing the routine was compiled. This option also arranges for the profile file to be produced automatically.

OPTIONS

—a all symbols are reported rather than just external symbols.

—l the output is sorted by symbol value.

—n the output is sorted by number of calls.

—s a summary profile file is produced in *mon.sum*. This is really only useful when more than one profile file is specified.

—v all printing is suppressed and a graphic version of the profile is produced on the standard output for display by the plot filters. When plotting, the numbers *low* and *high,* by default 0 and 100, may be given to cause a selected percentage of the profile to be plotted with accordingly higher resolution.

—z routines which have zero usage (as indicated by call counts and accumulated time) are nevertheless printed in the output.

FILES

mon.out	for profile
a.out	for namelist
mon.sum	for summary profile

CAVEATS

Prof may give quantization errors.

SEE ALSO

monitor(3c), profil(2), cc(1).

NAME

prompter — prompts the editor front end

SYNOPSIS

This program is not called directly but takes the place of an editor and acts as an editor front end.

prompter [–erase *c* **] [–kill** *c* **] [–editor** *editor* **]**
[–noeditor] [–help] *draftfile*

DESCRIPTION

Prompter is an editor for rapidly composing mail messages. It is an MH program in that it can have its own profile entry with switches, but it can't be invoked directly as all other MH commands can.

Prompter is normally called from *comp*, *repl*, *dist*, or *forw*, with a draft file as an argument. For example:

```
comp -editor prompter
```

calls **prompter** with the file *draft* already set up with blank components (such as To:, Cc:, etc). For each blank component **prompter** finds in the draft, it prompts you to enter a value for the component and accepts your response. Pressing ⟨RETURN⟩ causes the whole component to be left out of the message.

A '\' preceding a ⟨RETURN⟩ continues the response on the next line, allowing for multiline components. An ESC causes the component to be left in the output file, but to have an empty value. Any component that is non–blank is copied and echoed to your terminal.

If you don't specify any switches, **prompter** uses the editor specified by your EDIT environment variable, or if that is not set, */bin/xed* to edit the mail message.

Your *.mh_profile* can contain the following entries:

prompter–next:
To name the editor to be used on exit from **prompter**

Prompter has the following defaults:

—noeditor

OPTIONS

—editor *editor*
Enter *editor* after the components are entered.

—erase *c*
Use *c* as the erase character.

—help
Display a synopsis of the **prompter** command.

—kill *c*
Use *c* as the kill character.

—noeditor
Don't use an editor. You enter the message following the line of dashes and end it with an ⟨ESC⟩ or a ⟨Ctrl–D⟩. If the body is non–blank, the prompt is '–––––––Enter additional text'.

SEE ALSO

comp(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh).

NAME

prs — print an SCCS file

SYNOPSIS

prs [**—a**] [**—d**[*dataspec*]] [**—e**] [**—l**] [**—r**[*SID*]] files

DESCRIPTION

Prs ^ prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(5sccs)) in a user supplied format. If a directory is named, **prs** ^ behaves as though each file in the directory except non-SCCS files, were specified as a named file. If 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.

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(5sccs)) 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**.

OPTIONS

—a Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(1sccs)) and existing, i.e., delta type = *D*, deltas. If the **—a** keyletter is not specified, information for existing deltas only is provided.

—d*dataspec*
Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *prskeywords(5sccs)*) interspersed with optional user supplied text.

—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.

—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 .ft 2) of the most recently created delta is assumed.

EXAMPLES

```
prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file
```

may produce on the standard output:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

```
prs -d"Newest delta for pgm :M:: :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:
b178-12345
b179-54321
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????? Temporary file for multi-column output

DIAGNOSTICS

Use *sccshelp*(1sccs) for explanations.

CAVEATS

Non-SCCS files and unreadable files are silently ignored.

SEE ALSO

admin(1sccs), delta(1sccs), get(1sccs), prskeywords(5sccs), rmdel(1sccs), sccshelp(1sccs), sccsfile(5sccs).

NAME

ps — process status

SYNOPSIS

ps [**-a**] [**-c**] [**-e**] [**-f** *swapfile*] [**-g**]
[**-k** *corefile*] [**-l**] [**-n** *namelist*] [**-p** *proclist*]
[**-s**] [**-t** *termlist*] [**-u**] [**-v**] [**-w**] [**-x**]

DESCRIPTION

Ps prints information about processes. Normally, only the current user's processes are included in the output (see the **—a** option described below).

All output formats include, for each process, the process id PID, the control terminal of the process TT, the cpu time used by the process TIME (this includes both user and system time), the state of the process STAT, and an indication of the COMMAND which is running.

The state is given by a sequence of four letters, e.g. "RWNA". The first letter indicates the runnability of the process: R for runnable processes, T for stopped processes, P for processes in page wait, D for those in disk (or other short term) waits, S for those sleeping for less than about 20 seconds, and I for idle (sleeping longer than about 20 seconds) processes. The second letter indicates whether a process is swapped out, showing W if it is, or a blank if it is loaded (in–core); a process which has specified a soft limit on memory requirements and which is exceeding that limit shows >; such a process is (necessarily) not swapped.

The third letter indicates whether a process is running with altered CPU scheduling priority (nice); if the process priority is reduced, an N is shown, if the process priority has been artificially raised then a '<' is shown; processes running without special treatment have just a blank. The final letter indicates any special treatment of the process for virtual memory replacement. Currently the possibilities are A standing for VA_ANOM, S for VA_SEQL and blank for VA_NORM. An A typically represents a lisp program in garbage collection; S is typical of large image processing programs which are using virtual memory to sequentially address voluminous data.

OPTIONS

—a Asks for information about all processes with terminals (ordinarily only one's own processes are displayed).

—c Prints the command name, as stored internally in the system for purposes of accounting, rather than the command arguments, which are kept in the process' address space. This is more reliable, if less informative, since the process is free to destroy the latter information.

—e Asks for the environment to be printed as well as the arguments to the command.

—f *swapfile*
Swapfile is used as the name of a swap file to use instead of the default */dev/drum*.

—g Asks for all processes. Without this option, **ps** only prints "interesting" processes. Processes are deemed to be uninteresting if they are process group leaders. This normally eliminates top–level command interpreters and processes waiting for users to login on free terminals.

—k *corefile*
Causes *corefile,* a core image of the kernel, to be used in place of */dev/kmem* and */dev/mem.* This is used for postmortem system debugging.

—l Asks for a long listing, with fields PPID, CP, PRI, NI, ADDR, SZ, RSS and WCHAN as described below.

—n *namelist*
The file *namelist* is used as the file containing the system's namelist, where kernel symbols are obtained. Otherwise, kernel symbols are obtained from the table */dev/cvt* (see cvt(4)).

—p *proclist*
The output is restricted to data about processes whose process ID's are given in *proclist. Proclist* is a list of process ID's separated from one another by a comma, or a list of process ID's enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

—s Adds the size SSIZ of the kernel stack of each process (for use by system maintainers) to the basic output format.

—t *termlist*
The output is restricted to data about process whose controlling tty's are given in *termlist. Termlist* is a list of tty's separated from one another by a comma, or a list of tty's enclosed in double quotes and separated from one another by a comma and/or one or more spaces. Tty's may be specified in one of two forms: the device's file name (e.g. tty04) or if the device's file name starts with *tty*, just the digit identifier (e.g. 04).

—u A user–oriented output is produced. This includes fields USER, %CPU, SZ, and RSS as described below.

—v A version of the output containing virtual memory statistics is output. This includes fields RE, SL, PAGEIN, SIZE, RSS, LIM, TSIZ, TRS, %CPU and %MEM, described below.

—w
Use a wide output format (132 columns rather than 80); if repeated, e.g. ww, use arbitrarily wide output. This information is used to decide how much of long commands to print.

—x Asks even about processes with no terminal.

Fields which are not common to all output formats:

USER name of the owner of the process

%CPU	cpu utilization of the process; this is a decaying average over up to a minute of previous (real) time. Since the time base over which this is computed varies (since processes may be very young) it is possible for the sum of all %CPU fields to exceed 100%.
NI	process scheduling increment (see *setpriority*(2))
SZ	virtual size of the process (in 1024 byte units)
RSS	real memory (resident set) size of the process (in 1024 byte units)
LIM	soft limit on memory used, specified via a call to *setrlimit*(2); if no limit has been specified then shown as *xx*
TSIZ	size of text (shared program) image
TRS	size of resident (real memory) set of text
%MEM	percentage of real memory used by this process.
RE	residency time of the process (seconds in core)
SL	sleep time of the process (seconds blocked)
PAGEIN	number of disk i/o's resulting from references by the process to pages not loaded in core
UID	numerical user–id of process owner
PPID	numerical id of parent of process
CP	short–term cpu utilization factor (used in scheduling)
PRI	process priority (non–positive when in non–interruptible wait)
ADDR	swap address of the process
WCHAN	event on which process is waiting (an address in the system), with the initial part of the address trimmed off e.g. 80004000 prints as 4000

F flags associated with process as in ⟨*sys/proc.h*⟩:

SLOAD	000001	in core
SSYS	000002	swapper or pager process
SLOCK	000004	process being swapped out
SSWAP	000008	save area flag
STRC	000010	process is being traced
SWTED	000020	another tracing flag
SULOCK	000040	user settable lock in core
SPAGE	000080	process in page wait state
SKEEP	000100	another flag to prevent swap out
SDLYU	000200	delayed unlock of pages
SWEXIT	000400	working on exiting
SPHYSIO	000800	doing physical i/o (bio.c)
SVFORK	001000	process resulted from vfork()
SVFDONE	002000	another vfork flag
SNOVM	004000	no vm, parent in a vfork()
SPAGI	008000	init data space on demand from inode
SANOM	010000	system detected anomalous vm behavior
SUANOM	020000	user warned of anomalous vm behavior
STIMO	040000	timing out during sleep
SDETACH	080000	detached inherited by init
SOUSIG	100000	using old signal mechanism

A process that has exited and has a parent, but has not yet been waited for by the parent is marked ⟨defunct⟩; a process which is blocked trying to exit is marked ⟨exiting⟩. **Ps** makes an educated guess as to the file name and arguments given when the process was created by examining memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

FILES

/dev/cvt	table of kernel symbols
/dev/kmem	kernel memory
/dev/drum	swap device
/dev	searched to find swap device and tty names

DIAGNOSTICS

The diagnostices produced by **ps** are intended to be self–explanatory.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Things can change while **ps** is running; the picture it gives is only a close approximation to reality.

For compatibility, the "old" **ps** syntax is also supported. This syntax is

ps [acegklsuvwx[# ¦t*n*] [namelist [corefile [swapfile]]]]

SEE ALSO

kill(1), w(1), cvt(4).

NAME

ptog — **plot** to GPS filter

SYNOPSIS

ptog [*file* ...]

DESCRIPTION

Ptog transforms **plot** or **tplot** commands into a GPS. Input is taken from *file* if given, else from the standard input. Output is to the standard output.

Plot is a graphics utility and library that is part of 4.2 BSD UNIX. In System V UNIX, it is known as **tplot**. Neither is included with the current release of UTek. **Ptog** is included in the UTek Graphics Tools for compatibility with these other versions of UNIX.

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

cd, chdir, pushd, popd, dirs — directory change commands (**csh** *built-in)*

SYNOPSIS

cd [*dirname*]
or
cd *+n*

chdir [*dirname*]
or
chdir *+n*

pushd [*dirname*]
or
pushd *+n*

popd [*+n*]

dirs

DESCRIPTION

Csh maintains the name current directory and a directory stack which can be used to keep up with all of the places the user has been and may wish to go back to. With this, the user does not have to remember where to go back to when sidetracked.

The command **cd,** and its synonym **chdir,** change the current working directory of the shell. If no argument is given, the directory is changed to the user's home directory. If *dirname* begins with '/', './', or '../', an attempt is made to change to that directory. Otherwise, the directory *dirname* is searched for in the current directory and in each element of the variable *cdpath.* If this fails, and there is a variable named the same as the value of *dirname* whose value begins with a '/', that directory is used (see EXAMPLES). When the directory is changed, the top element of the directory stack is replaced by the new directory. When the *cdpath* or directory name variable is used to change the directory, the path of the new current directory is printed. With the argument *+n,* where *n* is a number (beginning at 0), the *n*'th element of the directory stack is moved from that position to the top of the stack. The contents of the directory stack is always printed upon completion.

The command **pushd** is used to add directory names to the directory stack, and to edit the stack, as well as change the current directory. With no arguments, **pushd** exchanges the top two elements of the directory stack. With a directory name argument, the directory is changed, and the name of the new current directory is pushed on to the stack. With the argument *+n,* where *n* is a number (beginning at 0), the *n*'th element of the directory stack is moved from that position to the top of the stack. The contents of the directory stack is always printed upon completion.

The command **popd** is used to remove elements from the directory stack. With no arguments, the top element of the stack is removed, resulting in changing the current working directory. With the argument *+n,* where *n* is a number (numbers begin at 0, but '+0' is not valid), the *n*'th element of the directory stack is removed. In the latter use, the current directory is unchanged. The contents of the directory stack is always printed upon completion.

The command **dirs** prints the contents of the directory stack in order. The top element (which is the current directory) is printed first.

The variable *cdpath* contains a list of directories to search if the directory name given is not a subdirectory of the current directory. This variable is maintained along with the environment variable CDPATH, which is used by the *sh(1sh)* **cd** command, but these variables are not the same. In **csh,** the current directory is implicitly the first element in *cdpath,* whereas in **sh,** the current directory must be given explicitly. In order to cope with this difference, the value of CDPATH is imported upon startup of a new shell. CDPATH is changed when *cdpath* is changed (using *set(1csh)),* but *cdpath* is not changed when CDPATH is changed (using *setenv(1csh)).*

The variable *cwd* is set by **csh** whenever the current directory is changed. Due to symbolic links and the ditributed file system, this variable may not always contain correct or desirable data. If the value of *cwd* is required to be correct, the variable *hardpaths* may be set, which causes all changes of directory to get the current directory path by calling *getwd(3c),* which will give the correct path. This makes directory changes somewhat slower. An alternate method is to use the command *pwd(1)* to get the correct current directory path when needed.

EXAMPLES

This example shows a use of the directory name variable.

```
set default=/
set cdpath=( ~ ~/* )
cd default
```

In this case, if there is no directory named 'default' in the current directory, the user's home directory, and any subdirectories of the user's home directory, the current directory will become '/'.

The following example shows some of the features of manipulation of the directory stack. Here, the character '%' represents the **csh** prompt.

```
% dirs
~

% pushd /bin
/bin ~
```

```
% pushd /etc
/etc /bin ~
% cd /usr
% dirs
/usr /bin ~
% pushd +2
~ /usr /bin
% popd +2
~ /usr
% popd
/usr
```

VARIABLES

CDPATH	The directory change search path.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the error message occurred.

CAVEATS

Shell scripts should never change to a subdirectory and attempt to go back by executing **cd .. ** , since the directory changed to may be a symbolic link to another directory whose parent directory is different from where the last **cd** was executed. At the very least, the *cwd* variable's value should be saved and used to go back.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1sh), chdir(1sh), continue(1csh), csh(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), chdir(2).

NAME

pwd — working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

EXAMPLES

To print the pathname of the current working directory type the following.

```
pwd
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

The length of the path returned is limited to MAXPATHLEN characters.

Pwd is actually a shell script which uses the **pwd** command built in to *sh*.

SEE ALSO

cd(1sh), pwd(1sh), sh(1sh), getwd(3c).

NAME

pwd — working directory name (*sh* built-in)

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

EXAMPLES

To print the pathname of the current working directory type the following.

```
pwd
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

The length of the path returned is limited to MAXPATHLEN characters.

The command **pwd** is also available as a shell script for programs that need to execute it directly, such as *csh*.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), chdir(2), execve(2), getwd(3c).

NAME

pwr — print current working root directory

SYNOPSIS

pwr [**—h**] [**—l**]

DESCRIPTION

Pwr prints the name of the current working root directory. When working on another machine via the distributed file system, the root directory of the current directory is not '/'. Instead, it is '//machine/', where 'machine' is the name of the machine that the current directory resides on.

Because of the way the distributed file system works, full pathnames which do not begin with a machine name are taken from the local (or originating) host. This command makes it easy to determine the full pathname of a file on the remote machine.

By default, if the current directory resides on the local machine, **pwr** will print '/'.

OPTIONS

—h Print only the name of the current machine. This option implies the **—l** option.

—l Print the full name of the root directory ('//machine/'), even if it is on the local machine.

EXAMPLES

To list the contents of the directory '/bin' on the current machine, the following command may be used.

```
ls `pwr`bin
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

pwd(1), pwd(1sh), getwd(3c), getwroot(3c).

NAME

pxref — Pascal cross–reference program

SYNOPSIS

pxref [—] *filename*

DESCRIPTION

Pxref makes a line numbered listing and a cross–reference of identifier usage for the program in *filename*. The optional '—' argument suppresses the listing. The keywords **goto** and **label** are treated as identifiers for the purpose of the cross–reference. **Include** directives are not processed, but cause the placement of an entry indexed by '#include' in the cross–reference.

CAVEATS

Identifiers are trimmed to 18 characters.

SEE ALSO

cxref(1).

NAME

qmod — modify, hold, or delete a queued MDQS request

SYNOPSIS

qmod [**—a** *atime*] [**—c** *count*] [**—f** *form*] [**—h**] [**——h**] [**—k**] [**—m**] [**——m**] [**—p** *priority*] [**—q** *queue*] *[user.]job# ...*

DESCRIPTION

Qmod modifies or deletes MDQS requests. Only request parameters that are common to all types of requests can be changed.

The command *qstat(1mdqs)* is used to obtain request identification. To specify a request submitted by the current user, it is necessary only to specify the job number. Only the superuser, the mdqs user, or a member of the systems group can modify a request submitted by other than the current user; in this case he or she must specify a request in the form *user.job#*.

OPTIONS

—a *atime*
: Changes the start time on the job to the time specified. It is illegal to change the start time to sometime earlier than the submit time. See *getdate(5mdqs)* for a description of time formats.

—c *copies*
: Changes the number of copies requested to *copies*.

—f *forms*
: Changes the forms for the request to the new value, *forms*.

—h
: Places a hold to be put on the specified requests. A held request keeps its place in the queue; another job can jump ahead of the held request if the held request is at the beginning of the queue.

——h
: Removes a hold from the specified requests.

—k
: Causes the requests specified to be killed (deleted).

—m
: Adds the notification upon completion option to the requests.

——m
: Removes mail notification upon completion.

—p *priority*
: Changes the priority of requests. Only the superuser, the mdqs user or a member of the systems group may request a priority of 0.

—q *queue*
: Changes the queue for the specified request to the new queue specified. The request is then sorted into the queue in accordance with its submit time and priority.

EXAMPLES

The first example will kill job 5 for the current user. The second example will change the forms requested for jobs 6 and 7 to wide paper, assuming either a system administrator or *jondoe* is submitting the modification. Request id's (for example, *jondoe.5*, *jondoe.6*, and *jondoe.7*) are obtained from the program **qstat**.

```
qmod -k 5
```

qmod -f wide jondoe.6 jondoe.7

FILES

etc/mdqsd	MDQS daemon
/usr/spool/q	Top of spooling directory tree
/etc/qconf	Configuration information for MDQS
/usr/lib/mdqs/forms	List of available forms

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

qstat(1mdqs), forms(5mdqs), mdqsd(8mdqs), lprm(1mdqs).

NAME

qsort — quick sort

SYNOPSIS

qsort [**—c***n*] [*vector ...*]

DESCRIPTION

Output is a vector of the elements from the input *vector* in ascending order. If no *vector* is given, the standard input is assumed.

OPTIONS

—c*n*

n is the number of output elements per line.

EXAMPLES

The following example outputs the the elements of A in ascending order.

```
qsort -c3 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

qstat — MDQS local and remote status program

SYNOPSIS

qstat [**—a**] [**—l**] [**—q** *queue*] [**—s**] [**—v**]

DESCRIPTION

The **qstat** program is used to display the status of the MDQS queues. The options allow for levels of verbosity and the ability to restrict the query to a particular queue. **Qstat** will report if the daemon is not running.

OPTIONS

—a List all queues and devices, including empty queues, remote queues, and the delayed queue. List specified queue, possible associated remote queue, and possibly remote associated devices if used in conjunction with the **—q** option.

—l Long format, giving request id, queue, requested forms, priority, byte size of job, and requested activation time.

—q *queue*
Only requests from the specified *queue* are listed. If the queue is mapped to a remote queue, and the **—a** Option is also specified, the remote queue will also be displayed. Only devices associated with the specified queue are displayed.

—s Slow search. Normally **qstat** will use some special files written out by the daemon to get the sorted queue information. If the daemon is not running, this option may be used so the queue directory will be searched by brute force.

—v Produces the most verbose listing.

EXAMPLES

```
qstat -l
```

will print all requests using the long format.

FILES

/etc/mdqsd	MDQS daemon
*/usr/spool/q/**	Spooling directories
/etc/qconf	Configuration information for MDQS

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Rsh privileges are required in order to access information for remote queues.

Things can change while **qstat** is running; the picture it gives is only a close approximation of reality. For instance, **qstat** may produce false error messages if it cannot find a particular file or if a data structure it is looking at changes underneath it.

SEE ALSO

qmod(1mdqs), mdqsd(8mdqs), lpq(1mdqs).

NAME

quit — terminate session

SYNOPSIS

quit

DESCRIPTION

Quit logs you off from a *graphics(1g)* shell. (It only works if your login shell is *sh(1sh)*.) To leave the **graphics** shell and return to your original shell, if any, type ⟨CTRL–D⟩ (or your *eof* character, if different).

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), sh(1sh), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

rand — generate random sequence

SYNOPSIS

rand [—**c***n*] [—**h***n*] [—**l***n*] [—**m***n*] [—**n***n*] [—**s***n*]

DESCRIPTION

Output is a vector of *number* random elements in a given range. The range can be set by specifying its low and high ends, or its low end and a multiplier.

The range of numbers is determined by the parameters *low*, *high*, and *multiplier*. The sequence of numbers is determined by the parameter *seed*. These parameters are set by command–line options.

Random numbers are first generated in the range 0 to 1, initialized by the *seed*. If a *multiplier* is given, each number is then multiplied by its value. If no *multiplier* is given, each number is multiplied by *high–low* (default 1). Finally, *low* is added to each number.

OPTIONS

—c*n*
n elements per output line.

—h*n*
high: = *n*. If not given, *high*: = 1.

—l*n*
low: = *n*. If not given, *low*: = 0.

—m*n*
multiplier: = *n*. If not given, *multiplier*: = *high–low*. If you use both **–m** and **–h**, the **–h** option is ignored.

—n*n*
number: = *n*. If not given, *number*: = 10.

—s*n*
seed: = *n*. If not given, *seed*: = 1. *n* must be an integer. Any given *seed* produces the same sequence of random numbers each time you use it.

EXAMPLES

The following example generates ten random numbers between 0 and 1.

```
rand
```

The following example generates ten random numbers between 10 and 35, three per line.

```
rand -l10,h35,c3
```

The following example also generates ten random numbers between 10 and 35, three per line.

```
rand -l10,m25,c3
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

rank — rank of vector

SYNOPSIS

rank [*vector* ...]

DESCRIPTION

Output is the number of elements in each input *vector*. If no *vector* is given, the standard input is assumed.

EXAMPLES

The following example outputs the number of elements in A.

```
rank A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

ranlib — convert archives to random libraries

SYNOPSIS

ranlib *archive ...*

DESCRIPTION

Ranlib converts each *archive* to a form which the loader can load more rapidly. **Ranlib** does this by adding a table of contents called *_.SYMDEF* to the beginning of the archive. **Ranlib** uses *ar(1)* to reconstruct the archive, so that sufficient temporary file space must be available in the file system which contains the current directory.

Ranlib works on both long and short format archive files.

Ranlib is automatically executed by **ar** when the contents of an archive are modified.

EXAMPLES

A typical invocation on the archive file *arch* follows:

```
ranlib arch
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Because generation of a library by **ar** and randomization of the library by **ranlib** are separate processes, phase errors are possible. The loader, **ld**, fails when the modification date of a library is more recent than the creation date of its dictionary; but this means that it fails even if you only copy the library.

SEE ALSO

ar(1), ld(1), nm(1), ar(5).

NAME

ratfor — rational FORTRAN dialect

SYNOPSIS

ratfor [**—h**][**—C**][**—6**][*filename*...]

DESCRIPTION

Ratfor converts a rational dialect of FORTRAN into ordinary irrational FORTRAN. **Ratfor** provides control flow constructs essentially identical to those in C:

Statement grouping:

```
{ statement; statement; statement }
```

Decision–making:

```
if (condition) statement [ else statement ]
switch (integer value) {
case integer: statement
[ default: ] statement
}
```

Loops:

```
while (condition) statement
for (expression; condition; expression) statement
do limits statement
repeat statement [ until (condition) ]
break
next
```

Free form input:

```
multiple statements/line; automatic continuation
```

Comments:

```
# this is a comment
```

Translation of relationals:

```
>, >=, etc., become .GT., .GE., and so forth.
```

Return (expression)

```
returns expression to caller from function
```

Define:

```
define name replacement
```

Include:

```
include filename
```

Ratfor is best used with *efl(1)* and *f77(1)*.

OPTIONS

—h Converts a string '...' to *27h*....

—C

Generates comments in the output.

—6 Sets continuation field to 6, the default being 1 for UTek. It may be followed by a character to be used as the continuation character. If none is given the default is **&**.

SEE ALSO

f77(1).

NAME

rcp — remote file copy

SYNOPSIS

rcp *filename1 filename2*
rcp [**—r**] [**—l**] *filename ... directory*

DESCRIPTION

Rcp copies files between machines. Each *filename* or *directory* argument is either a remote filename of the form *rhostname:pathname* , or a local filename (containing no colon (:) characters, or a slash (/) before any colons (:)).

If the **—r** is specified and any of the source files are directories (and **—l** is not specified, or the file is not a symbolic link), **rcp** copies each subtree rooted at that name; in this case the destination must be a directory. With **—l** any source file which is a symbolic link will be copied without being followed, whether or not it links to a directory, and whether or not **—r** is given.

If *path* is not a full pathname, 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. Therefore, access to the remote system is based on the contents of either of two files on the remote system. Those files are *.rhosts* in a remote user's login directory or */etc/hosts.equiv*. See *rlogin(1n)* for more details. **Rcp** normally requires the current local user name to exist on *rhostname* . To circumvent this requirement, hostnames may take the form *rhostname.rname* to use *rname* rather than the current user name on the remote host.

Rcp handles third party copies, where neither source nor target files are on the current machine.

OPTIONS

—r Recursively copy all files and subdirectories from the given source directories. With **—r,** the destination must be a directory.

—l Copy symbolic links instead of following them. Copied links will look just like the original.

EXAMPLES

Example:

```
rcp host1:"*.c" src
```

In the above example, **rcp** copies all files with extensions *.c* in the user's login directory on *host1* to the directory named *src* on the local system. Note that **.c* is quoted so that the local shell will not attempt to expand the '*'. The quotes will, however, be stripped off by the local shell so that the shell on the remote host will do the filename expansion.

Example:

```
rcp -r host1.jeffm:doc /usr/chrisd
```

This example copies the directory *doc* (in *jeffm*'s home directory on *host1*) and all the files in it (including other directories) to *chrisd*'s directory on the local host. The use of '*.jeffm*' is only necessary if the user initiating the **rcp** is not *jeffm*. For this approach to work, the user initiating the **rcp** must appear along with the name of his host machine in the *.rhosts* file in *jeffm*'s home directory. (See *.rhosts(5n)*).

Example:

```
rcp -r file1 file2 host2:file3 host1:dir1
```

This example copies the local files *file1* and *file2* and remote file *file3* on *host2* to the directory *dir1* on *host1*.

DIAGNOSTICS

Rcp prints error messages received from the remote **rcp** as well as messages generated by the shell on the remote host.

The latter have the format:

`rcp : Message from` *rhost:* ' ⟨message⟩ '

If you are using *csh(1csh)* on the remote system then **rcp** will print any output generated by commands in the remote *.cshrc* file.

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[INTERNAL]	An unexpected error occurred. Execution was terminated. Record the message and save the core file for analysis. Contact service personnel at your Tektronix field office.

SEE ALSO

ftp(1n), rsh(1n), rlogin(1n). .rhosts(5n), hosts.equiv(5n).

NAME

rcs — change RCS file attributes

SYNOPSIS

rcs [—**A***oldfilename*] [—**L**] [—**N***name[:rev]*] [—**P**] [—**U**]
[—**a***logins*] [—**c***string*] [—**e***logins*] [—**i**] [—**l***rev*]
[—**n***name[:rev]*] [—**o***range*] [—**q**] [—**s***state[:rev]*]
[—**t***txtfilename*] [—**u***[rev]*] *filename ...*

DESCRIPTION

Rcs creates new RCS files or changes attributes of existing ones. An RCS file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For **rcs** to work, your login name must be on the access list. This doesn't apply if the access list is empty, you are the owner of the file or the superuser, or the **-i** option is present.

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

OPTIONS

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

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

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

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

You are prompted to enter descriptive text as described in *ci(1rcs)*.

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

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

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

—q Quiet mode. Diagnostics are not printed.

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

—t [*txtfilename*]
Writes descriptive text into the RCS file (deletes the existing text). If *txtfilename* is omitted, **rcs** prompts you for text supplied from the *standard* input, terminated with a line containing a single dot (.) or ⟨CTRL–D⟩. Otherwise, the descriptive text is copied from the file *txtfilename*. If the **–i** option is present, descriptive text is requested even if **–t** is not given. The prompt is suppressed if the *standard* input is not a terminal.

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

—A*oldfilename*
Appends the access list of *oldfilename* to the access list of the RCS file.

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

—N*name*[*:rev*]
Same as **–n**, except that it overrides a previous assignment of *name*.

—P
Causes the access and modification dates of the RCS file to stay the same before and after modification, unless the current revision was outdated. See CAVEATS.

—U

Sets locking to *nonstrict*. Nonstrict locking means that the owner of a file need not lock a revision for checkin. This option should NOT be used for files that are shared. The default is strict locking, but this may be changed by setting the environment variable **RCSLOCK** to nonstrict.

EXAMPLES

The following example unlocks the current revision of the file *example.c* if it is locked (if it is not locked, a message to that effect is printed):

```
rcs -u example.c
```

FILES

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

,RCSt$$	Temporary storage for data during changes. $$ is the current process id.
,,*	The semaphore file. This file prevents other **rcs** updates to the file.

DIAGNOSTICS

The RCS filename and the revisions outdated are written to the diagnostic output.

VARIABLES

RCSLOCK	If set to nonstrict, all new RCS files will have locking set to nonstrict mode. Otherwise, locking is strict.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

The maximum number of revisions that can be stored in a single RCS file is 719. When there are more than 700 revisions in a file, a warning message is printed on the terminal (if possible) every time an RCS command works on the file. See the manual page for *rcsfile(5rcs)* for information on what action to take in this case.

The maximum length of a description message (see **—i**) is 2048 characters. Longer messages are truncated.

On older versions of RCS, the maximum number of revisions that can be stored in a single RCS file is 239. No warning is displayed on the terminal if this number is exceeded.

The **—P** option is supplied so that **make** does not think that a file has been changed just because the administrative information was modified. This causes **make** to do less work, but it can cause problems if the values of any of the RCS keyword values are used. For example, if a revision logging system is used to store the version numbers of source files into the object code, these numbers may not be correct if the **—P** option is used.

SEE ALSO

ci(1rcs), co(1rcs), ident(1rcs), rlog(1rcs), rcsdiff(1rcs), rcsintro(1rcs), rcsmerge(1rcs), rcsfile(5rcs).

NAME

rcsdiff — compare RCS revisions

SYNOPSIS

rcsdiff [**—b**] [**—cefhn**] [**—r***rev1*] [**—r***rev2*] *filename ...*

DESCRIPTION

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

The options **–b**, **–c**, **–e**, **–f**, and **–h** have the same effect as described in *diff(1)*; option **–n** generates an edit script of the format used by RCS.

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

If *rev1* is given, but *rev2* is omitted, **rcsdiff** compares revision *rev1* of the RCS file with the contents of the corresponding working file.

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

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

OPTIONS

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

—c Produces a **diff** with lines of context. The default is to present three lines of context and may be changed (for example, to 10 by **—c10**). With **—c** the output format is modified slightly: the output beginning with identification of the files involved and their creation dates and then each change is separated by a line with a dozen *'s. The lines removed from *filename1* are marked with a dash (–); those added to *filename2* are marked with a plus sign (+). Lines which are changed from one file to the other are marked in both files with an exclamation point (!).

—e Produces a script of *a, c* and *d* commands for the editor **ed**, which will recreate *filename2* from *filename1*. 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
```

Extra commands are added to the output when comparing directories with **—e,** so that the result is a *sh(1sh)* script for converting text files which are common to the two directories from their state in *dir1* to their state in *dir2*.

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

—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.

—n Generates an edit script of the format used by RCS.

—r*rev1*
Revision *rev1* is the first revision used in the compare. Defaults to the latest revision on the trunk.

—r*rev2*
Revision *rev2* is the second revision used in the compare. Defaults to the current working version.

EXAMPLES

The following command shows differences between the file *f.c* and the current revision of the RCS file *f.c,v*:

```
rcsdiff f.c
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

Rcsdiff uses modified versions of *diff(1)* and *diff3(1)*, which must be maintained separately. These programs are contained in the directory */usr/lib*, and are called *rdiff* and *rdiff3*.

The maximum number of revisions that can be stored in a single RCS file is 719. When there are more than 700 revisions in a file, a warning message is printed on the terminal (if possible) every time an RCS command works on the file. See the manual page for *rcsfile(5rcs)* for information on what action to take in this case.

On older versions of RCS, the maximum number of revisions that can be stored in a single RCS file is 239. No warning message is displayed on the terminal if this number is exceeded.

SEE ALSO

ci(1rcs), co(1rcs), diff(1), ident(1rcs), rlog(1rcs), rcs(1rcs), rcsintro(1rcs), rcsmerge(1rcs), rcsfile(5rcs).

NAME

rcsin — see whether RCS files are checked in or not

SYNOPSIS

rcsin [**—d** *path*] [**—p** *path*] [**—s**] [**—v**] [*file ...*]

DESCRIPTION

If any *file* specified is a working file with a locked RCS file, **rcsin** prints to standard output the name of the working file. If *file* is an RCS file with a writable working file, **rcsin** prints the same information. If **rcsin** prints any messages, it also exits with a nonzero status.

File can be the name of either an RCS file (ending in *,v*) or a working file -- **rcsin** derives one name from the other. If *file* is omitted, all RCS files in the current directory and the RCS subdirectory are examined (for example, **,v RCS/*,v*). By using the **—d** and **—p** options, the directories to be searched can be set. This is useful in cases where RCS files are being obtained from various directories.

Rcsin is often used within makefiles to assure that all work-in-progress is checked in. It can be used to make sure that writable working files are not deleted by **make.**

OPTIONS

—d *path*
Directory. In addition to searching . and *./RCS* , **rcsin** looks in the directories listed in *path* for the RCS file *(*,v)*. *Path* can be the name of one directory or a list of directories separated by whitespace.

—p *path*
Path. **Rcsin** searches the directories in *path* for the RCS file. *Path* can be the name of one directory or a list of directories separated by whitespace.

—s Silent. No messages are written to standard-out; only an exit status is returned.

—v Verbose. **Rcsin** prints the reason *file* is not checked in (for example, "is writable" or "is locked") and, in the case of a locked file, either prints the version number and locker for each locked revision or prints the fact that the RCS lock file (*,*,*) exists.

EXAMPLES

The following example checks to see if the file *foo.d* is checked out. If not, **rcsin** returns 0. If it is checked out, **rcsin** tells you why (it is either a *writable file* or *checked out*).

```
rcsin -v foo.d
```

DIAGNOSTICS

Exit status is 0 if all went well, 1 if any *file* is locked or any associated working files are writable, and 2 if some other error occurred.

CAVEATS

If you use **—p** and **—d** together, the order of the arguments determines the results. The **—p** option removes whatever was in the search path,

while **—d** appends to whatever was in the search path.

The *path* argument to the **—d** and **—p** options contains directory names separated by whitespace. This means that a directory name containing whitespace can not be used with these options.

SEE ALSO

ci(1rcs), co(1rcs), ident(1rcs), make(1), rcs(1rcs), rlog(1rcs), rcsfile(5rcs).

NAME

rcsintro — introduction to RCS commands

DESCRIPTION

The Revision Control System (RCS) manages multiple revisions of text files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently (for example: programs, documentation, graphics, papers, form letters, and so forth).

The basic user interface is extremely simple. The novice only needs to learn two commands: **ci** and **co. Ci**, short for *checkin*, deposits the contents of a text file into an archival file called an RCS file. An RCS file contains all revisions of a particular text file. **Co**, short for *checkout*, retrieves revisions from an RCS file.

Functions of RCS

- Storage and retrieval of multiple revisions of text. RCS saves all old revisions in a space efficient way. Changes no longer destroy the original, because the previous revisions remain accessible. Revisions can be retrieved according to ranges of revision numbers, symbolic names, dates, authors, and states.
- Maintenance of a complete history of changes. RCS logs all changes automatically. Besides the text of each revision, RCS stores the author, the date and time of checkin, and a log message summarizing the change. The logging makes it easy to find out what happened to a module, without having to compare source listings or having to track down colleagues.
- Resolution of access conflicts. When two or more programmers wish to modify the same revision, RCS alerts the programmers and prevents one modification from corrupting the other.
- Maintenance of a tree of revisions. RCS can maintain separate lines of development for each module. It stores a tree structure that represents the ancestral relationships among revisions.
- Merging of revisions and resolution of conflicts. Two separate lines of development of a module can be coalesced by merging. If the revisions to be merged affect the same sections of code, RCS alerts the user about the overlapping changes.
- Release and configuration control. Revisions can be assigned symbolic names and marked as released, stable, experimental, and so forth. With these facilities, configurations of modules can be described simply and directly.
- Automatic identification of each revision with name, revision number, creation time, author, and so forth. The identification is like a stamp that can be embedded at an appropriate place in the text of a revision. The identification makes it simple to determine which revisions of which modules make up a given configuration.

- Minimization of secondary storage. RCS needs little extra space for the revisions (only the differences). If intermediate revisions are deleted, the corresponding deltas are compressed accordingly.

Getting Started with RCS

Suppose you have a file *f.c* that you wish to put under control of RCS. Invoke the checkin command

```
ci  f.c
```

This command creates the RCS file *f.c,v*, stores *f.c* into it as revision 1.1, and deletes *f.c*. It also asks you for a description. The description should be a synopsis of the contents of the file. All later checkin commands will ask you for a log entry, which should summarize the changes that you made.

Files ending in *,v* are called RCS files (*v* stands for *versions*); the others are called working files. To get back the working file *f.c* in the previous example, use the checkout command

```
co  f.c
```

This command extracts the latest revision from *f.c,v* and writes it into *.f.c*. You can now edit *f.c* and check it back in by invoking

```
ci  f.c
```

Ci increments the revision number properly. If **ci** complains with the message

```
ci error: no lock set by <your login>
```

then your system administrator has decided to create all RCS files with the locking attribute set to strict. In this case, you should have locked the revision during the previous checkout. Your last checkout should have been

```
co  -l  f.c
```

Of course, it is too late now to do the checkout with locking, because you probably modified *f.c* already, and a second checkout would overwrite your modifications. Instead, invoke

```
rcs  -l  f.c
```

This command will lock the latest revision for you, unless somebody else got ahead of you already. In this case, you'll have to negotiate with that person.

Locking assures that you, and only you, can check in the next update, and avoids nasty problems if several people work on the same file. Even if a revision is locked, it can still be checked out for reading, compiling, and so forth. All that locking prevents is a CHECKIN by anybody but the locker.

If your RCS file is private, for example, if you are the only person who is going to deposit revisions into it, strict locking is not needed and you can turn it off. If strict locking is turned off, the owner off the RCS file need not have a lock for checkin; all others still do. Turning strict locking off and on is done with the commands

```
rcs -U f.c      and      rcs -L f.c
```

If you don't want to clutter your working directory with RCS files, create a subdirectory called RCS in your working directory, and move all your RCS files there. RCS commands will look first into that directory to find needed files. All the commands discussed above will still work, without any modification. (Actually, pairs of RCS and working files can be specified in 3 ways: (a) both are given, (b) only the working file is given, (c) only the RCS file is given. Both RCS and working files may have arbitrary path prefixes; RCS commands pair them up intelligently).

To avoid the deletion of the working file during checkin (in case you want to continue editing), invoke

```
ci -l f.c      or      ci -u f.c
```

These commands check in *f.c* as usual, but perform an implicit checkout. The first form also locks the checked in revision, the second one doesn't. Thus, these options save you one checkout operation. The first form is useful if locking is strict, the second one if not strict. Both update the identification markers in your working file (see below).

You can give **ci** the number you want assigned to a checked in revision. Assume all your revisions were numbered 1.1, 1.2, 1.3, and so forth, and you would like to start release 2. The command

```
ci -r2 f.c      or      ci -r2.1 f.c
```

assigns the number 2.1 to the new revision. From then on, **ci** will number the subsequent revisions with 2.2, 2.3, and so forth. The corresponding **co** commands

```
co -r2 f.c      and      co -r2.1 f.c
```

retrieve the latest revision numbered 2.*x* and the revision 2.1, respectively. **Co** without a revision number selects the latest revision on the trunk (for example, the highest revision with a number consisting of two fields). Numbers with more than two fields are needed for branches. For example, to start a branch at revision 1.3, invoke

```
ci -r1.3.1 f.c
```

This command starts a branch numbered 1 at revision 1.3, and assigns the number 1.3.1.1 to the new revision. For more information about branches, see *rcsfile(5rcs)*.

Automatic Identification

RCS can put special strings for identification into your source and object code. To obtain such identification, place the marker

```
$Header$
```

into your text (for instance, inside a comment). RCS will replace this marker with a string of the form

```
$Header: filename revision_number
date time author state $
```

With such a marker on the first page of each module, you can always see with which revision you are working. RCS keeps the markers up to date automatically. To propagate the markers into your object code, simply put them into literal character strings. In C, this is done as follows:

```
static char rcsid[] = "$Header$";
```

The command **ident** extracts such markers from any file, even object code and dumps. Thus, **ident** lets you find out which revisions of which modules were used in a given program.

You may also find it useful to put the marker `$Log$` into your text, inside a comment. This marker accumulates the log messages that are requested during checkin. Thus, you can maintain the complete history of your file directly inside it. There are several additional identification markers; see *co(1rcs)* for details.

CAVEATS

The maximum number of revisions that can be stored in a single RCS file is 719. When there are more than 700 revisions in a file, a warning message is printed on the terminal (if possible) every time an RCS command works on the file. See the manual page for *rcsfile(5rcs)* for information on what action to take in this case.

SEE ALSO

ci(1rcs), co(1rcs), ident(1rcs), merge(1rcs), rlog(1rcs), rcs(1rcs), rcsdiff(1rcs), rcsmerge(1rcs), rcsfile(5rcs).

NAME

rcsmerge — merge RCS revisions

SYNOPSIS

rcsmerge [**—p**] **—r***rev1* [**—r***rev2*] *filename*

DESCRIPTION

Rcsmerge incorporates the changes between *rev1* and *rev2* of an RCS file into the corresponding working file. If **–p** is given, the result is printed on the *std.* output; otherwise the result overwrites the working file.

A filename ending in *,v* is an RCS filename; otherwise it is considered a working filename. **Rcsmerge** derives the working filename from the RCS filename and vice versa, as explained in *co(1rcs)*. A pair consisting of both an RCS and a working filename may also be specified.

Rev1 cannot be omitted. If *rev2* is omitted, the latest revision on the trunk is assumed. Both *rev1* and *rev2* may be given numerically or symbolically.

Rcsmerge prints a warning if there are overlaps, and delimits the overlapping regions as explained in **co –j**. The command is useful for incorporating changes into a checked–out revision.

OPTIONS

—p Results are printed on *std.* output.

—r*rev1*
Revision *rev1* is the first revision used in the merge.

—r*rev2*
Revision *rev2* is the second revision used in the merge. Defaults to the latest revision on the trunk.

EXAMPLES

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

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

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

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

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

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

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

FILES

/tmp/d3[abc]$$	Temporary files for merging. $$ is current process id.
,RCSt1$$	Temporary storage for revisions being merged.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

Rcsmerge does not work for files that contain lines with a single dot (.).

Rcsmerge uses *merge(1rcs),* so modifications to **merge** may result in problems with **rcsmerge.**

The maximum number of revisions that can be stored in a single RCS file is 719. When there are more than 700 revisions in a file, a warning message is printed on the terminal (if possible) every time an RCS command works on the file. See the manual page for *rcsfile(5rcs)* for information on what action to take in this case.

On older versions of RCS, the maximum number of revisions that can be stored in a single RCS file is 239. No warning message is displayed on the terminal if this number is exceeded.

SEE ALSO

ci(1rcs), co(1rcs), ident(1rcs), merge(1rcs), rlog(1rcs), rcs(1rcs), rcsdiff(1rcs), rcsintro(1rcs), rcsfile(5rcs).

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

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

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

FILES

/tmp/d3[abc]$$	Temporary files for merging. $$ is current process id.
,RCSt1$$	Temporary storage for revisions being merged.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

Rcsmerge does not work for files that contain lines with a single dot (.).

Rcsmerge uses *merge(1rcs),* so modifications to **merge** may result in problems with **rcsmerge.**

The maximum number of revisions that can be stored in a single RCS file is 719. When there are more than 700 revisions in a file, a warning message is printed on the terminal (if possible) every time an RCS command works on the file. See the manual page for *rcsfile(5rcs)* for information on what action to take in this case.

SEE ALSO

ci(1rcs), co(1rcs), ident(1rcs), merge(1rcs), rlog(1rcs), rcs(1rcs), rcsdiff(1rcs), rcsintro(1rcs), rcsfile(5rcs).

NAME

read — read a line from standard input (**sh** built–in)

SYNOPSIS

read *name ...*

DESCRIPTION

One line is read from the standard input; successive words of the input are assigned to the *names* in order, with the leftover words to the last variable. The return code is 0 unless the end–of–file is encountered.

EXAMPLES

The following shell script reads lines from the standard input and echos them until a line is encountered whose first word is *end* or the end of input is reached.

```
#!/bin/sh
while read first other
do
        echo "$first" "$other"
        if test "end" = "$first"
        then
                exit
        fi
done
```

RETURN VALUE

[0] Data was read as requested.

[1] End–of–file has been encountered.

CAVEATS

It is not possible to redirect the input of **read** in the current shell. For example, the command **read f** *<filename* waits for input and then prints the message **illegal io**.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), line(1), login(1), pwd(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2).

NAME

readonly — sets variables to readonly (*sh* built–in)

SYNOPSIS

readonly [*name* . . .]

DESCRIPTION

The specified names are marked *read only* and the values of these names may not be changed by subsequent assignment. If no arguments are given, a list of all **readonly** names is printed. This is especially useful in a restricted shell environment.

EXAMPLES

The following command marks the variables **HOME** and **PATH** as read only. Any subsequent assignment to either of these will result in an error.

```
readonly HOME PATH
```

RETURN VALUE

[NO_ERRS] Command completed without error.

CAVEATS

The read only attribute for variables applies only to variables in the current shell. Exported variables do not retain read only status.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2).

NAME

red — restricted text editor

SYNOPSIS

red [—] [**-p** [*prompt*] [**-q**] [**—x**] [*filename*]

DESCRIPTION

Red is a restricted version of *ed(1)*. It is used in conjunction with the restricted mode of *sh(1sh)*.

The differences between **red** and **ed** are that **red** will only allow the user to edit files in the current directory, and will not allow execution of shell commands using the **!** command. Attempts to bypass these restrictions results in an error message saying that the shell is restricted.

See the documentation for **ed** for a description of the commands.

OPTIONS

— Suppresses the printing of character counts by **e**, **r**, and **w** commands, of diagnostics from **e** and **q** commands, and of the **!** prompt after a **!** shell command.

-p [*prompt*]
Causes the specified *prompt* to be printed when **ed** has finished with a command and is waiting for the next one. If no *prompt* is specified, * is used. Without the **—p** option, no prompting is done.

—q Allows the signal SIGQUIT (normally generated by the character ⟨\⟩) to terminate the edit session, and turns off the — option. Normally, SIGQUIT is ignored.

—x An **X** command is simulated first to handle an encrypted file.

EXAMPLES

The following input causes the @ prompt to be printed when **ed** is done with a command:

```
red -p@
```

FILES

/tmp/e#	Temporary; # is the process number.
ed.hup	Work is saved here if the terminal is hung up.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

SEE ALSO

ed(1), grep(1), sed(1), sh(1sh), stty(1).

NAME

refile — file message(s) in (an)other folder(s)

SYNOPSIS

refile [-ask] [-noask] [[-file *file*] ¦ [*msgs*]]
[-help] [-link] [-nolink] [-preserve]
[-nopreserve] [-src +*folder* **]** +*folder* ...

DESCRIPTION

Refile moves or links mail messages from one mail folder into one or more other mail folders (see *mv(1)* and *ln(1)*). The folder that the messages come from is called the source folder, while the other folders are called destination folders. When a message is filed, it is linked into the destination folder(s) if possible, and is copied otherwise.

If the destination folders are all on the same file system, multiple filing causes little storage overhead. This is a good way to cross-file or multiply-index messages.

If a destination folder doesn't exist, **refile** asks if you want to create one. If you respond with *no* the **refile** is aborted.

Your *.mh_profile* can contain the following entries:

`Path:` To determine the user's MH directory
`Current-Folder:` To find the default current folder
`Folder-Protect:` To set mode when creating a new folder

Refile has the following defaults:

—ask
—nolink
—nopreserve
msgs defaults to the current message
—src +*folder* defaults to the current folder

You can use the keyword **all** as the *msgs* argument to **refile** all the mail messages in a mail folder. See the EXAMPLES section.

OPTIONS

—ask
If the destination folder doesn't exist, **refile** asks you if you want to create the folder.

—noask
If the destination folder doesn't exist, **refile** creates it without asking for confirmation.

—file *file*
Move or link *file* into the destination folder. Note that *file* doesn't have to be in a mail folder; it can be in any directory.

—help
Print a message showing how to use **refile**.

—link
Preserve the mail message in the source folder. (it does a link, *ln(1)*, rather than a move, *mv(1)*). If neither **—link** nor the keyword **all** are specified, the last message specified becomes the current message. Otherwise, the current message is left unchanged.

—nolink
Delete the filed messages from the source folder after copying them to the destination folder.

—preserve
Overrides the message renaming conventions. Normally, when a message is filed, it is assigned the next highest number available in each of the destination folders. With this option naming conflicts may occur, so use this switch cautiously. (See *pick(1)* for more details on message numbering.)

—src *+folder*
Take the specified mail messages from *folder*. *Folder* becomes the current folder for future MH commands.

EXAMPLES

If a message is received from Jones about the ARPA Map Project, the following instruction allows the message to be found in either of the two folders *jones* or *Map*:

```
refile cur +jones +Map
```

The following example moves all the mail messages from the folder *inbox* to the folder *savebox*:

```
refile all -nolink -src +inbox +savebox
```

FILES

$HOME/.mh_profile The user profile

SEE ALSO

comp(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh).

NAME

regcmp — regular expression compile

SYNOPSIS

regcmp [—] *filename ...*

DESCRIPTION

Regcmp, in most cases, precludes the need for calling *regcmp(3pw)* from C programs. This saves on both execution time and program size. The command **regcmp** compiles the regular expressions in *filename* and places the output in *filename*.**i**. If the — option is used, the output will be placed in *filename*.**c**. The format of entries in *filename* 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. *Filename*.**i** files may thus be **included** into C programs, or *filename*.**c** files may be compiled and later loaded. In the C program which uses the **regcmp** output, **regex** *c,line)* will apply the regular expression named *abc* to *line*. Diagnostics are self–explanatory.

EXAMPLES

The following is an example of the contents of *filename*:

```
name "([A—Za—z][A—Za—z0—9_]*)$0"
telno "\({0,1}([2—9][01][1—9])$0\){0,1} *"
```

In the C program that uses the **regcmp** output,

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

will apply the regular expression named *telno* to *line.*

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

awk(1), egrep(1), fgrep(1), grep(1), sed(1), lex(1), regcmp(3pw), regex(3c).

NAME

rehash, unhash, hashstat — hashing control and report commands (**csh** built-in)

SYNOPSIS

rehash
unhash
hashstat

DESCRIPTION

In order to speed up execution path searching, *csh(1csh)* hashes the commands in each of the directories in the execution path. When executing a command, the command name is hashed to see if it could possibly be in a given directory. If the hash value of the command name indicates that the command may be in a given directory, **csh** attempts to execute that command. If the execution succeeds, the operation is called a *hit*. If not, the operation is called a *miss*.

The command **rehash** causes the internal hash table to be recomputed. This is usually required when new commands are added to directories in the execution path.

The command **unhash** disables execution path hashing. This is useful when a lot of changes are being made to the directories in the execution path.

The command **hashstat** prints a short report about the efficiency of the hashing. The report is of the form: *number* **hits,** *number* **misses,** *number* **%.** The hits and misses numbers correspond to the number of hits and misses (as explained above) since the shell was invoked. The percentage given is the percentage of the total attempts that succeeded in a hit.

EXAMPLES

In order to cause the internal command hash table to be recomputed, execute the command

```
rehash
```

RETURN VALUE

[NO_ERRS] Command completed without error.

CAVEATS

The more commands that exist in a given directory, the more times the hashing result in a miss.

The hashing in **csh** is quite different from *sh(1sh)* hashing, in that the latter stores information on a command when it is executed and is more efficient when a directory contains a lot of commands.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh),

exit(1csh), fg(1csh), glob(1csh), goto(1csh), hash(1sh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), execve(2).

NAME

remcom — remove comments

SYNOPSIS

remcom [*file* ...]

DESCRIPTION

Remcom copies its input to its output with comments removed. Comments are as defined in C (i.e., /* *comment* */). Input is from *file*(*s*) if given, else from the standard input.

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

renice — alter priority of running processes

SYNOPSIS

/etc/renice *priority* [[**—p**] *pid ...*] [**—g** *pgrp ...*] [**—u** *user ...*]

DESCRIPTION

Renice alters the scheduling priority of one or more running processes. The *pid, pgrp,* and *user* parameters are process ID's, process group ID's, or user names. Executing **renice** on a process group causes all processes in the process group to have their scheduling priority altered. (The process group ID is the process ID of the parent process.) Executing **renice** on a user causes all processes owned by the user to have their scheduling priority altered. By default, the processes to be affected are specified by their process ID's. To force parameters to be interpreted as process group ID's, a **—g** may be specified. To force the parameters to be interpreted as user names, a **—u** may be given. Supplying **—p** will reset parameters to (the default) process ID's.

Users other than the superuser can only alter the priority of processes they own, and can only monotonically increase their *nice value* within the range 0 to PRIO_MIN (20). (This prevents overriding administrative fiats.) The superuser can alter the priority of any process and set the priority to any value in the range PRIO_MAX (—20) to PRIO_MIN (20). Useful priorities are: 19 (the affected processes will run only when nothing else in the system wants to); 0 (the *base* scheduling priority); and anything negative (to make things go very fast).

OPTIONS

—g *pgrp*
Interpret arguments up to the next option as process group ID's.

—p *pid*
Interpret arguments up to the next option as process ID's.

—u *user*
Interpret arguments up to the next option as user ID's.

EXAMPLES

The following example changes the priority of process ID's 987 and 32, and all processes owned by the user's daemon and root.

```
/etc/renice +1 987 -u daemon root -p 32
```

FILES

/etc/passwd	Used to map user names to user ID's

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

If you make the priority very negative, then the process cannot be interrupted. To regain control, you make the priority greater than zero.

Non superusers can not increase scheduling priorities of their own processes, even if they were the ones that decreased the priorities in the first place.

SEE ALSO

nice(1), getpriority(2), setpriority(2).

NAME

repeat — repeatedly execute command (**csh** built–in)

SYNOPSIS

repeat *count command* [*args...*]

DESCRIPTION

The **repeat** command executes the given *command* with its arguments *count* times. The *count* must be an integer or a variable expansion which expands to a number. The *command* must be a simple command, not an alias. Redirection is done exactly once, even if *count* is 0.

EXAMPLES

The following command line builds a file called 'output' which consists of four copies of the contents of the file 'input'.

```
repeat 4 cat input > output
```

RETURN VALUE

The return value is the return value of the last command executed. If the command is not found or an error occurred,

CAVEATS

Quoted arguments containing the filename metacharacters '*', '?', and '[...]' do not work correctly, in that the metacharacters are expanded the second time through the loop. For example, if the current directory contains the files 'hello', 'bye', and 'seeya', the command

```
repeat 4 echo '*'
```

results in the output

```
*
bye hello seeya
bye hello seeya
bye hello seeya
```

instead of four lines of the character '*'.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), xargs(1).

NAME

repl — reply to a message

SYNOPSIS

repl [+*folder*][*msgs*][**—editor** *editor*][**—inplace**][**—annotate**] [**—help**][**—noinplace**][**—noannotate**][**—cc**][**—nocc**]

DESCRIPTION

Repl produces a reply to an existing mail message. In its simplest form (with no arguments), **repl** sets up a message-form skeleton in reply to the current message in the current folder, invokes the editor, and sends the composed message if so directed. The composed message is constructed as follows:

```
To: ⟨Reply-To⟩ or ⟨From⟩
Subject: Re: ⟨Subject⟩
In-reply-to: Your message of ⟨Date⟩
        ⟨Message-Id⟩
```

where field names enclosed in angle brackets (⟨ ⟩) indicate the contents of the named field from the message to which the reply is being made. If **—cc** is specified, a `Cc:` line is constructed from contents of the `Cc:` and/or `To:` fields from the message to which the reply is being made. Once the skeleton is constructed, an editor is invoked (as in **comp**, **dist**, and **forw**). While in the editor, the message being replied to is available through a link called `@`. In **ex**, this means the replied-to message may be *read* with **:r** `@`.

As in **comp**, **dist**, and **forw**, you are queried before the message is sent. If the **—annotate** switch is specified, the replied-to message will be annotated with the single line, as such:

Replied: ⟨⟨*Date Time*⟩⟩

only if the message is sent directly. The command **comp —use** may be used to pick up interrupted editing, as in **dist** and **forw**; the **—inplace** switch annotates the message in place, so that all folders with links to it will see the annotation.

Your *.mh_profile* can contain the following entries:

`Path:` To determine the user's MH directory
`Editor:` To override use of */etc/mh/prompter* as the default editor
`Current-Folder:` To find the default current folder

Repl has the following defaults:

+*folder* defaults to current
msgs defaults to cur
—editor defaults to */etc/mh/prompter*
—noannotate
—noinplace
—cc

If a *+folder* is specified, it will become the current folder, and the current message will be set to the replied–to message.

FILES

$HOME/.mh_profile	The user profile.
⟨mh-dir⟩/draft	The constructed message file.
/usr/bin/send	To send the composed message.

SEE ALSO

comp(1mh), fmt(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh), mh_profile(5mh).

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 and then to the shell, because <CR> often doesn't work; often none of this will echo.

It isn't a bad idea to follow **reset** with *tset(1)*.

CAVEATS

Doesn't set tabs properly; it cannot intuit personal choices for interrupt and line–kill characters, so it leaves these 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.

SEE ALSO

stty(1), tset(1).

NAME

return — exit from a function (**sh** built–in)

SYNOPSIS

return [*n*]

DESCRIPTION

Return causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

OPTIONS

n The return value.

EXAMPLES

The following function returns the number of arguments it was given:

```
args () {
        return $#
}
```

RETURN VALUE

The return value is either the value given by the argument *n* or the return value of the last command executed.

CAVEATS

Since functions are executed in the current shell, functions should only use **exit** if they cause the current shell to exit.

SEE ALSO

alias(1csh), break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2).

NAME

rev — reverse lines of a file

SYNOPSIS

rev [*filename...*]

DESCRIPTION

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

EXAMPLES

In the following example, the lines in the file *xxx* will be reversed and written on standard output:

```
rev xxx
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Lines containing more than 256 characters will be truncated.

SEE ALSO

tail(1).

NAME

rlog — print log messages and other information about RCS files

SYNOPSIS

rlog [—**thc**] [—**L***[lockers]*] [—**R**] [—**d***dates*] [—**l***[lockers]*]
[—**r***revisions*]
[—**s***states*] [—**w***[logins]*] *filename* . . .

DESCRIPTION

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

Rlog with no options prints the following information for each RCS file: full pathname of the RCS file, working filename, head (for example, the number of the latest revision on the trunk), access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for printing, and descriptive text. This is followed by entries for revisions in reverse chronological order for each branch. For each revision, **rlog** prints revision number, author, date/time, state, number of lines added/deleted (with respect to the previous revision), locker of the revision (if any), and log message.

OPTIONS

—**c** Prints only the head (current revision number) of the named files. This option is overridden by the —**h** and —**t** options.

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

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

—**l**[*lockers*]
Prints information about locked revisions. If the comma-separated list *lockers* of loginnames is given, only the revisions locked by the given loginnames are printed. If the list is omitted, all locked revisions are printed.

—**r***revisions*
Prints information about revisions given in the comma-separated list *revisions* of revisions and ranges. A range *rev1—rev2* means revisions *rev1* to *rev2* on the same branch, *—rev* means revisions from the

beginning of the branch up to and including *rev*, and *rev*— means revisions starting with *rev* to the end of the branch containing *rev*. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range.

—s*states*
Prints information about revisions whose state attributes match one of the states given in the comma–separated list *states*.

—t Prints the same as **–h**, plus the descriptive text.

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

—L*[lockers]*
Print information only for files without strict locking and no locked revisions. If the comma–separated list *lockers* of loginnames is given, the file is only chosen if it has revisions locked by one or more of the given names.

—R
Print only the pathname of the RCS file.

Combinations of the options **–d**, **–l**, **–r**, **–s**, and **–w** print the intersection of the revisions selected by each option. For these options, **rlog** also prints the information provided by **–t**.

EXAMPLES

The following input prints information about all revisions of the file *example.c* between revisions 1.3 and 1.8 which are currently locked:

```
rlog -r1.3-1.8 -l example.c
```

This next example shows a use for the **—c** flag. This command forces the current revision of the file *example.c* to be unlocked. This is useful if someone else has the revision locked.

```
rcs -u`rlog -c example.c` example.c
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

The maximum number of revisions that can be stored in a single RCS file is 719. When there are more than 700 revisions in a file, a warning

message is printed on the terminal (if possible) every time an RCS command works on the file. See the manual page for *rcsfile(5rcs)* for information on what action to take in this case.

SEE ALSO

ci(1rcs), co(1rcs), ident(1rcs), rcs(1rcs), rcsdiff(1rcs), rcsintro(1rcs), rcsmerge(1rcs), rcsfile(5rcs).

NAME

rlogin — remote login

SYNOPSIS

rlogin *rhostname* [**—e***c*] [**—l** *username*] [**—8**]
rhostname [**—e***c*] [**—l** *username*] [**—8**]

DESCRIPTION

Rlogin connects your terminal on the current local host system to the remote host, *rhostname*.

Each host has a file */etc/hosts.equiv* (see *hosts.equiv(5n)*) which contains a list of host names which are 'trusted' enough that their users are allowed to run programs on the local system (see *rsh(1n)*), copy files between the local and remote sytems (*see rcp(1n)*), and login to the local system, all without the use of passwords. Each user may also have a private equivalence list in a file *.rhosts* in his or her 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 not listed in */etc/hosts.equiv* or the appropriate *.rhosts* file, then a login and password will be prompted for on the remote machine as in *login(1)*. To avoid some security problems, the *.rhosts* file must be owned by either the remote user or root and may not be a symbolic link.

If there is no home directory for the user on the remote system, **rlogin** will print `Sorry, home directory required` and prompt for another login.

The second form of invoking **rlogin**, where you omit the "rlogin" is possible if you place in some directory in your search path a symbolic link (see *ln(1)*) linking *rhostname* to */bin/rsh*. This works because *rsh* can inspect its invocation, and if it finds only a hostname, *rsh* assumes you want to login in.

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

⟨RETURN⟩ ⟨TILDE⟩ ⟨DOT⟩ ⟨RETURN⟩

disconnects from the remote host, where tilde (~) is the escape character. A different escape character may be specified by the **—e** option.

If you are a *csh(1)* user, you can suspend a remote login by either

⟨RETURN⟩ ⟨TILDE⟩ ⟨CTRL–Z⟩ ⟨RETURN⟩

or

⟨RETURN⟩ ⟨TILDE⟩ ⟨CTRL–Y⟩ ⟨RETURN⟩

The former will stop all output until you bring the **rlogin** job back into the foreground, the latter will not. However, if you want the output from a suspended rlogin (or any other suspended job, for that matter) not to be held up, you must also turn off the *stty(1)* option *tostop*, i.e. stty –tostop. Typically you would place that *stty* option in your *.login*.

Note that you can have a whole chain of **rlogin**'s and selectively suspend to any previous one. Suppose you logged into machine *able*, and from there **rlogin**'d to *baker*, and from there you **rlogin**'d to *charlie*. If you currently are on *charlie* and want to suspend that job and return to *able*, you could type

⟨RETURN⟩ ⟨TILDE⟩ ⟨CTRL–Z⟩ ⟨RETURN⟩.

You could also elect to suspend your *charlie* **rlogin** and return to *baker* by typing

⟨RETURN⟩ ⟨TILDE⟩ ⟨TILDE⟩ ⟨CTRL–Z⟩ ⟨RETURN⟩.

Having said this it should be noted that it is much more efficient to **rlogin** direct to a host, rather than set up a chain of **rlogin**s.

OPTIONS

—e*c*
Set the escape character to be the *c* character. There is no space separating this option flag and the argument character.

—l *username*
Specify the login *username* to login as on the remote host.

—8 Set rlogin to transparently handle 8–bit characters. You lose flow control with this option (cntl–S and cntrl–Q) but you gain the ability to transfer 8–bit data. Typically this would be useful if your terminal sends 8 bit graphical data in response to some command from the host.

EXAMPLES

This example illustrates how to **rlogin** to another user's account (in this case, *sheryl*'s) on remote host *engr1*:

```
rlogin engr1 -l sheryl
```

If there was an entry in the *.rhosts* file in *sheryl*'s home directory for the user attempting the **rlogin**, then no password would be required.

FILES

*/usr/hosts/** For *rhost* version of the command.

RETURN VALUE

[3] Remote process has died, Connection is closed.

[USAGE] Incorrect command line syntax. Execution terminated.

[NP_WARN] An error warranting a warning message occurred. Execution continues.

[NP_ERR] An error occurred that was not a system error. Execution terminated.

[P_ERR] A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

[P_WARN] A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

More terminal characteristics should be propagated.

SEE ALSO

rsh(1n), hosts.equiv(5n), .rhosts(5n).

NAME

rm — remove (unlink) files or directories

SYNOPSIS

rm [**—f**] [**—i**] [**—r**] [—] *filename*...

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 or is currently busy (see CAVEATS), 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 and no errors are reported when the **—f** (force) option is given.

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.

The null option — indicates that all the arguments following it are to be treated as filenames. This allows the specification of filenames starting with a minus.

It is forbidden to remove the file '..' merely to avoid the antisocial consequences of inadvertently doing something like **rm**—r.*.

If no filenames are given, a usage message is printed, unless the **—f** option is given.

OPTIONS

—f No questions are asked and no errors are reported when the **—f** (force) option is given.

—i **Rm** asks whether to delete each file, and, under **—r**, whether to examine each directory.

—r **Rm** recursively deletes the entire contents of the specified directory, and the directory itself.

— Indicates that all the arguments following it are to be treated as filenames. This allows the specification of filenames starting with a minus.

EXAMPLES

The following example removes the contents of the directory */usr/example* and each of its subdirectories, and the directory itself. It will not complain about files for which the user does not have write permission.

```
rm -rf /usr/example
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

When a file is busy (being executed), **rm** will ask if the file is to be removed, even if the permissions are such that the user has write permission. This applies to all users, including the superuser.

Questions are printed on standard error. Therefore, when redirecting standard error to a file, the **—f** option should be used.

Recursive removes of directory structures of more than NOFILE – 3 levels do not fail. In effect, **rm** can remove any depth of directory structure.

SEE ALSO

rmdir(1), rmdir(2), unlink(2).

NAME

rmail — handle remote mail received via uucp

SYNOPSIS

rmail *user ...*

DESCRIPTION

Rmail interprets incoming mail received via *uucp(1n),* collapsing "From" lines in the form generated by *mail(1mh)* into a single line of the form "return–path!sender", and passing the processed mail on to *sendmail(8mh).*

Rmail is explicitly designed for use with **uucp** and **sendmail.**

CAVEATS

Rmail should not reside in /bin.

SEE ALSO

mail(1mh), uucp(1n), sendmail(8mh).

NAME

rmdel, rmchg, cdc — remove a delta from an SCCS file

SYNOPSIS

rmdel —r*SID filename...*

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 .ft 2) specified must *not* be that of a version being edited for the purpose of making a delta (for example, if a *p-filename* (see *get(1sccs)*) exists for the named SCCS file, the .ft 2) specified must *not* appear in any entry of the *p-filename*).

If a directory is named, **rmdel** behaves as though each file in the directory were specified as a named file. If a name of — (a dash) 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.

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, the permissions are: if you make a delta you can remove it; or if you own the file and directory you can remove a delta.

OPTIONS

—r*SID*

Uniquely identifies which delta is to be removed.

EXAMPLES

The following example removes 1.3 delta from the file *s.foo.c*:

```
rmdel -r1.3 foo.c
```

FILES

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

DIAGNOSTICS

Use *sccshelp(1sccs)* for explanations.

CAVEATS

Non–SCCS files and unreadable files are silently ignored.

SEE ALSO

delta(1sccs), get(1sccs), sccshelp(1sccs), prs(1sccs), sccsfile(5sccs).

NAME

rmdir — remove (unlink) directories or files

SYNOPSIS

rmdir *directory . . .*

DESCRIPTION

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

The current directory (.) cannot be removed even if it is empty.

EXAMPLES

The following example removes the directories *Mail* and *Other* from the current directory. Both directories must be empty.

```
rmdir Mail Other
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

rm(1), rmdir(2), unlink(2).

NAME

rmf — remove folder

SYNOPSIS

rmf [*+folder*][**—help**]

DESCRIPTION

Rmf removes all of the files (messages) within the specified (or default) folder, and then removes the directory (folder). If there are any files within the folder which are not a part of MH, they will *not* be removed, and an error will be produced. If the folder is given explicitly or the current folder is a subfolder (for example, a selection list from pick), it will be removed without confirmation. If no argument is specified and the current folder is not a selection–list folder, the user will be asked for confirmation.

Rmf irreversibly deletes messages that don't have other links, so use it with caution.

If the folder being removed is a subfolder, the parent folder will become the new current folder, and **rmf** will produce a message telling the user this has happened. This provides an easy mechanism for selecting a set of messages, operating on the list, then removing the list and returning to the current folder from which the list was extracted (See the example under *pick(1mh)*).

The files that **rmf** will delete are *cur* (current), any file beginning with a comma, and files with purely numeric names. All others will produce error messages.

Rmf of a read–only folder will delete the *cur*— entry from the profile without affecting the folder itself.

Your *.mh_profile* can contain the following entries:

```
Path: To determine the user's MH directory
Current-Folder: To find the default current folder
```

Rmf has the following defaults:

+folder defaults to *current*, usually with confirmation

Rmf will set the current folder to the parent folder if a subfolder is removed; or, if the current folder is removed, it will make *inbox* current. Otherwise, it doesn't change the current folder or message.

OPTIONS

—help

Displays a synopsis of the **rmf** command.

EXAMPLES

This example removes the folder *Check* and all the MH files under it. *Check* has three files under it that are not a part of MH; these are not removed.

```
rmf +Check
```

FILES

$HOME/.mh_profile	The user profile

SEE ALSO

comp(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh).

NAME

rmm — remove messages

SYNOPSIS

rmm [*+folder*][*msgs*][**—help**]

DESCRIPTION

Rmm removes the specified messages.

The current message is not changed by **rmm**, so a **next** will advance to the next message in the folder as expected.

Your *.mh_profile* can contain the following entries:

`Path:` To determine the user's MH directory
`Current-Folder:` To find the default current folder

Rmm has the following defaults:

+folder defaults to current
msgs defaults to cur

If a folder is given, it will become current.

OPTIONS

—help
Displays a synopsis of the **rmm** command.

EXAMPLES

This input removes the message *xxx* from the folder *Check*. *Check* becomes current.

```
rmm +Check
```

FILES

$HOME/.mh_profile The user profile

SEE ALSO

comp(1mh), fmt(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), mh(1mh), mhl(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh), mh(5mh), mh_profile(5mh).

NAME

root — root function

SYNOPSIS

root [—c*n*] [—r*n*] [*vector* ...]

DESCRIPTION

Output is a vector with each element being the *root* root of the corresponding element from the input *vector(s)*. If no *vector* is given, the standard input is assumed.

OPTIONS

—c*n*

n is the number of output elements per line.

—r*n*

root: = *n*. If not given, *root*: = 2.

EXAMPLES

The following example outputs the 3.5th root of each element of A, three per line.

```
root -p3.5,c3 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

round — rounded value

SYNOPSIS

round [—**c***n*] [—**p***n*] [—**s***n*] [*vector* ...]

DESCRIPTION

Output is the rounded value for each element of the input *vector(s)*. If no *vector* is given, the standard input is assumed.

OPTIONS

—**c***n*

n is the number of output elements per line.

—**p***n*

n is the number of places following the decimal point in the rounded value. *n* is an integer in the range 0 to 9, 0 by default.

—**s***n*

n is the number of significant digits rounded to. *n* is an integer in the range 0 to 9, 9 by default.

EXAMPLES

The following example outputs the value of each element of A rounded to two significant digits, three per line.

```
round -s2,c3 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

rsh — remote shell

SYNOPSIS

rsh *host* [**—l** *username*] [**—n**] *command*
host [**—l** *username*] [**—n**] *command*

DESCRIPTION

Rsh connects to the specified *host,* and executes the specified *command.* **Rsh** 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; **rsh** 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 local machine, while quoted metacharacters are interpreted on the remote machine.

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 are also commands in the directory */usr/hosts*; if you put this directory in your search path then the **rsh** can be omitted.

OPTIONS

—l*username*
Specify a different remote username.

—n Redirect the input of **rsh** to */dev/null* so that **rsh** will not block on *reads* from the terminal. See CAVEATS below.

EXAMPLES

The following command appends the remote file *file0* on host *ecs* to *file1* on the local machine:

```
rsh ecs cat file0 >> file1
```

However, this next command appends *file0* on *ecs* to *file1* which is also on *ecs*:

```
rsh ecs cat file0 ">>" file1
```

Remember that characters that are special to the shell (like '>', '<', '?', '[', ']', '*', '¦', etc.) must be quoted to be passed to the shell running on the remote system.

Suppose the name of a host on which you wish to execute commands is *ecs*. Then if you link a host name to **rsh** as in the following:

```
ln -s /bin/rsh /usr/hosts/ecs
```

and add */usr/hosts* to your path search path, then you can execute a remote command without specifying '**rsh**', for example:

```
ecs cat file0 >> file1
```

FILES

/etc/hosts	Contains hostnames: standard and nicknames.
*/usr/hosts/**	Contains hostnames for local machines.

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

If you are using *csh(1csh)* and put a *rsh(1n)* in the background without redirecting its input away from the terminal, it will block even if no *reads* are posted by the remote command. If no input is desired you should redirect the input of **rsh** to */dev/null* using the **—n** option.

You cannot run an interactive command (like *vi(1)*); use *rlogin(1n)*.

Stop signals stop the local **rsh** process only.

SEE ALSO

rlogin(1n), hosts.equiv(5n), .rhosts(5n).

NAME

ruptime — show status of hosts on local area network

SYNOPSIS

ruptime [**—a**][**—l**][**—t**][**—u**][**—U**][*hostname*]...

DESCRIPTION

Ruptime prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last one, five, and 15 minutes for each machine on the local network; status lines are formed from packets broadcast by each host on the network once a minute.

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

Users who are idle for an hour or more are not counted unless the **—a** flag is given.

If any host names are present on the command line, status lines will be printed only for those hosts.

OPTIONS

—a Count all users logged in, even if they have been idle for an hour or more.

—l Sort status lines by load average.

—t Sort status lines by length of time operational.

—u Sort status lines by number of users.

—U Generate a list of host systems that are up, one name to a line.

FILES

/vmunix	System name list.
*/usr/spool/rwho/whod.**	Data files.

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Use **uptime(1N)** to obtain statistics about other hosts. **Ruptime** is supplied for applications requiring compatibility with Berkeley's 4.2 Unix. Unfortunately the 4.2 approach of each host broadcasting a statistics packet every minute has two major disadvantages. It clogs the network with information that is only seldom used, and it burdens the workstation with processing all the broadcast packets. For this reason, **ruptime** and

rwho(1N) as well as the daemon **rwhod(8N)** are not normally operational on the network. If you wish to use them, they are located in */usr/old*. To install them, copy **rwho** and **ruptime** to */bin* and **rwhod** to */etc*. You will also have to enable running **rwhod** whenever the workstation is rebooted, by modifying the file */etc/rc.net*. The command lines to start **rwhod** are commented out in the *rc.net* file, so remove the comment characters ('#'), and **rwhod** will be started when the workstation is rebooted.

SEE ALSO

who(1N), uptime(1N), w(1).

NAME

who — who's logged in on machines on local area network

SYNOPSIS

rwho [**—a**] [*hostname*] . . .

DESCRIPTION

The **rwho** command produces output similar to **who**, for 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 no *hostname* is listed, **rwho** reports on all hosts. Otherwise, it will list the users only for those *hostnames* named.

OPTIONS

—a If a user hasn't typed to the system for an hour of more, then the user will be omitted from the output of **rwho** unless the **—a** flag is given.

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.

CAVEATS

The preferred method to list users on remote systems is the **who(1n)** command. If your application requires **rwho**, see **ruptime(1n)** for installation instructions.

Rwho can not list any more than 39 users from any host.

SEE ALSO

ruptime(1n), uptime(1n), who(1n).

NAME

scan — produce a one–line–per–message scan listing

SYNOPSIS

scan [*+folder*][*msgs*][**—ff**][**—header**][**—noff**][**—noheader**]

DESCRIPTION

Scan produces a one–line–per–message listing of the specified messages. Each **scan** line contains the message number (name), the date, the `From` field, the `Subject` field, and, if room allows, some of the body of the message.

If there is sufficient room left on the **scan** line after the subject, the line will be filled with text from the body, preceded by <<. **Scan** actually reads each of the specified messages and parses them to extract the desired fields. During parsing, appropriate error messages will be produced if there are format errors in any of the messages.

Your *.mh_profile* can contain the following entries:

`Path:` To determine the user's MH directory
`Current–Folder:` To find the default current folder

Scan has the following defaults:

+folder defaults to current
msgs defaults to all
—noff
—noheader

If a folder is given, it will become current. The current message is unaffected.

OPTIONS

—header
This switch produces a header line prior to the **scan** listing.

—ff
This switch will cause a form feed to be output at the end of the **scan** listing.

EXAMPLES

The following input:

```
scan +Train 15 16 18 19
```

gives these results:

```
15+ 7/ 5 Dcrocker nned <<Last week I asked some of
16 - 7/ 5 dcrocker message id format <<I recommend
18 7/ 6 Obrien Re: Exit status from mkdir
19 7/ 7 Obrien "scan" listing format in MH
```

The + on message 15 indicates that it is the current message. The — on message 16 indicates that it has been replied to, as indicated by a `Replied:` component produced by an **—annotate** switch to the **repl** command.

FILES

$HOME/.mh_profile The user profile

SEE ALSO

comp(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh).

NAME

sccsdiff — compare two versions of an SCCS file

SYNOPSIS

sccsdiff [—r*SID1*][—r*SID2*][—**p**][—s*n*] *filenames* . . .

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.

OPTIONS

—p Pipe output for each file through *pr(1).*

—r*SID?*

SID1 and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff(1sccs)* in the order given.

—s*n*

N is the file segment size that **bdiff** will pass to *diff(1).* This is useful when **diff** fails due to a high system load.

EXAMPLES

Sccsdiff will compare two versions of the file *example.x* (1.3 and 1.4) and pipe the output through *pr(1)*:

```
sccsdiff -r1.3 -r1.4 -p example.x
```

FILES

/tmp/get?????	Temporary files

DIAGNOSTICS

file : No differences	You will get this response if the two versions are the same.

Use *sccshelp(1sccs)* for explanations.

SEE ALSO

bdiff(1sccs), get(1sccs), sccshelp(1sccs), pr(1).

NAME

sccshelp, msghlp — ask for help about error messages

SYNOPSIS

sccshelp *args*
msghlp *args*

DESCRIPTION

Sccshelp finds information to explain a message from a command or explain the use of a command. One or more arguments may be supplied.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, or one of the following types:

type 1 — Begins with nonnumerics, ends in numerics. The nonnumeric prefix is usually an abbreviation for the program or set of routines which produced the message (for example, **sys13**, for message 13 from the **sys** command). The file */usr/lib/errtags/helploc* will be checked for a file corresponding to the nonnumeric prefix. That file will then be searched for the message. If */usr/lib/errtags/helploc* does not exist or the prefix is not found there, the search will be attempted on */usr/lib/errtags/⟨non-numeric prefix⟩*, (for example, */usr/lib/errtags/zz)*, with the search key being ⟨*remainder of arg*⟩ (for example, 32).

type 2 — Does not contain numerics (as a command, such as **get**) The file */usr/lib/errtags/cmds* is searched, with the search key being the whole argument.

type 3 — Is all numeric (for example, **212**), or if the file as determined above does not exist, the search will be attempted on /usr/lib/errtags/default with the search key being the entire argument.

The response of the program will be the explanatory information related to the argument, if there is any.

EXAMPLES

This request is of type 2 described above. **Sccshelp** will search the file */usr/lib/errtags/cmds* to find information that will explain the command **ident:**

```
sccshelp ident
```

FILES

/usr/lib/errtags — Directory containing files of message text.

/usr/lib/errtags/⟨non-numeric prefix⟩ — The file searched if the message is not found in */usr/lib/errtags/helploc.*

/usr/lib/errtags/default	The file searched if the argument is all numeric.
/usr/lib/errtags/helploc	The file searched if the argument has a nonnumeric prefix.
/usr/lib/errtags/cmds	Contains the syntax lines for the commands.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.

SEE ALSO

msghlp(1), ERROR(3c), errtag(5).

NAME

script — make typescript of terminal session

SYNOPSIS

script [**—a**] [**—e***c*] [**—8**] [*file*]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript (also called the *log*) is written to *file*, or appended to *file* if the **—a** option is given. If you don't specify a filename, the typescript is saved in the file *typescript*.

Script begins by executing the shell specified in your **SHELL** environment variable (*/bin/sh* if **SHELL** is not set). The typescript ends when you log out of the shell.

You can send commands directly to **script** by beginning a line with the command character (~ by default). These commands and their output are not put in the typescript file. Commands **script** recognizes are:

~**l**	Toggle logging off/on.
~^**Z**	Temporarily suspend **script** (*csh* only).
~**?**	Print a list of **script** commands.
~~	A single tilde.

OPTIONS

—a Output is appended to the output file.

—e*c*
Set the command character to be the *c* character.

—8 Use 8–bit characters. You lose flow control with this option (cntl–S and cntrl–Q) but you gain the ability to transfer 8–bit data. Typically this is used when your terminal sends 8 bit graphical data in response to commands from the host.

EXAMPLES

The following example appends the output from **script** to the file *outfile*.

```
script  -a  outfile
```

The following example changes the **script** command character to the **%** character. Output is put in the file named *typescript*.

```
script -e%
```

VARIABLES

SHELL	The shell to be executed. This is usually set to the value of the user's login shell.

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

CAVEATS

Script puts *everything* in the typescript file; including your input, prompts, all output from programs, and control characters.

SEE ALSO

csh(1csh), rlogin(1N), sh(1sh), tee(1).

NAME

sdb — symbolic debugger

SYNOPSIS

sdb [*objfil* [*corfil* [*directory*]]]

DESCRIPTION

Sdb is a symbolic debugger which can be used with C, PASCAL, and F77 programs. It may be used to examine their files and to provide a controlled environment for their execution.

Objfil is an executable program file which has been compiled with the **—go** (debug) on the C and F77 compilers or the **-g** (debug) option on the Pascal compiler. The default for *objfil* is *a.out*. *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is *core*. The core file need not be present. If *objfil* and *corfil* are both present, then *directory* may be used to specify the directory containing the source files for *objfil* . Otherwise the source files are assumed to be in the current directory.

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

Names of variables are written just as they are in C, PASCAL, or F77. Note that the F77 compiler converts names to lower case. Variables local to a procedure may be accessed using the form *procedure:variable*. If no procedure name is given, the procedure containing the current line is used by default. It is also possible to refer to structure members as *variable.member*, pointers to structure members as *variable*–*)member* and array elements as *variable[number]*. Combinations of these forms may also be used.

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

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

COMMANDS:

t Print a stack trace of the terminated or stopped program.

T Print the top line of the stack trace.

variable/lm

Print the value of variable according to length *l* and format *m*. If *l* and *m* are omitted, **sdb** chooses a length and format suitable for the variable's type as declared in the program. The length specifiers are:

b one byte
h two bytes (half word)
l four bytes (long word)
number
string length for formats **s** and **a**

Legal values for *m* are:

c character
d decimal
u decimal, unsigned
o octal
x hexadecimal
f 32 bit single precision floating point
g 64 bit double precision floating point
i machine instruction with symbolic information (constant address only)
I machine instruction without symbolic information (constant address only)
s Assume variable is a string pointer and print characters until a null is reached.
a Print characters starting at the variable's address until a null is reached.
p pointer to procedure

The length specifiers are only effective with the formats **d**, **u**, **o** and **x**. If one of these formats is specified and *l* is omitted, the length defaults to the word length of the host machine. The last variable may be redisplayed with the command dot–slash (*./*). ⟨CTRL–D⟩ will display the next 10 addresses following the address of the last variable displayed. This option is useful for dumping arrays and stacks.

Registers may also be dislayed by placing the register name followed by a **%** in place of *variable*.

The *sh(1sh)* metacharacters * and **?** may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, both variables local to the current procedure and global (common for F77) variables are matched, while if a procedure name is specified then only variables local to that procedure and matched. To match only global variables (or blank common for F77), the form *:pattern* is used. The name of a common block may be specified instead of a procedure name for F77 programs.

variable=lm
linenumber=lm
number=lm
procedure:=lm
Print the address of the variable,procedure or line number, or the value of the number in the specified format. If no format is given, then **lx** is used. The last variant of this command provides a

convenient way to convert between decimal, octal, and hexadecimal. If the variable is of type register the register number will be printed.

variable!*value*
Set the variable to the given value. The value may be a number, character constant, or a variable. If the variable is of type float or double, the value may also be a floating constant.

linenumber?
Disassemble the first machine instruction associated with *linenumber*. ⟨CTRL–D⟩ will disassemble the next 10 instructions following the last one disassembled.

COMMANDS FOR EXAMINING SOURCE FILES:

e*procedure*
e*filename.c*
Set the current file to the file containing the named procedure or the *filename*. Set the current line to the first line in the named procedure or file. All source files are assumed to be in *directory*. The default for *directory* is the working directory. If no procedure or filename is given, the current procedure and filenames are reported.

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

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

p Print the current line.

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

⟨CTRL–D⟩
Scroll. Print the next 10 lines. Set the current line to the last line printed.

w Window. Print the 10 lines around the current line.

number
Set the current line to the given linenumber. Print the new current line.

count +
Advance the current line by *count* lines. Print the new current line.

count –
Retreat the current line by *count* lines. Print the new current line.

COMMANDS FOR CONTROLLLING THE EXECUTION OF THE SOURCE PROGRAM:

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

linenumber **c** *count*
linenumber **C** *count*
Continue after a breakpoint or interrupt. If *count* is given, it specifies the number of breakpoints to be ignored. **C** continues with the signal which caused the program to stop and **c** ignores it.

If a linenumber is specified then a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted when the command finishes.

count **s**
Single step. Run the program through *count* lines. If no count is given then the program is run for one line.

count **S**
Single step, but step through subroutine calls.

count **i**
count **I**
Single step machine instruction. Run the program through *count* machine instructions. If no count is given then the program is run for one instruction. **I** reactivates the signal the program is stopped with and **i** ignores it.

k Kill the debugged program.

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

linenumber **b** *commands*
Set a breakpoint at the given line. If a procedure name without a linenumber is given (for example, *proc:*), a breakpoint is placed at the first line in the procedure even if it was not compiled with the debug flag. If no *linenumber* is given, a breakpoint is placed at the current line.

If no *commands* are given then execution stops just before the

breakpoint and control is returned to **sdb**. Otherwise the *commands* are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons.

constant **#**
(Pound sign) Set memory breakpoint at address and break if the memory **read/write** has occured. *Constant* is an integer constant representing an *address*. If **r** appears after the **#**, as in **#r**, then the memory breakpoint is set at the address specified and the break occurs on a memory **read**. If **w** appears after the **#**, then the memory breakpoint is set at the address and the break occurs on a memory **write**. If **d** appears after the **#** then the breakpoint (specified by address) is deleted.

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

B Print a list of the currently active breakpoints.

D Delete all breakpoints.

l Print the last executed line.

linenumber **a**
Announce. If *linenumber* is of the form *proc:number*, the command effectively does a *linenumber* **b l**. If *linenumber* is of the form *proc:*, the command effectively does a *proc:* **b T**.

MISCELLANEOUS COMMANDS:

variable$ **m**
Single step until variable changes. Variable can be in any specified active procedure. This will not work with register variables. It is also very, very slow.

v Toggle verbose mode. This causes the machine instructions to be printed as they are executed when single stepping with **s** or **S**.

n Toggle National–32000/Compiler disassembler format. This causes the assembler code to be displayed in National–32000 format (default) or Compiler format.

x Display contents of all machine registers.

X Display current value of **pc** and disassemble current instruction.

! *command*
The command is interpreted by *sh(1sh)*.

⟨*newline*⟩
If the previous command printed a source line then advance the

current line by one line and print the new current line. If the previous command displayed a core location then display the next core location.

" *string*
Print the given string.

q Exit the debugger.

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

V Print the version number.

Q Print a list of procedures and files being debugged.

Y Toggle debug output.

M Print locations of text and data segments in *object* and *core* files.

FILES

a.out	Default binary file
core	Default core image file

DIAGNOSTICS

Error reports are either identical to those of *adb(1)* or are self-explanatory.

CAVEATS

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

Arrays must be of one dimension and of zero origin to be correctly addressed by **sdb**.

The default type for printing F77 parameters is incorrect. Their address is printed instead of their value.

Tracebacks containing F77 subprograms with multiple entry points may print too many arguments in the wrong order, but their values are correct.

Sdb understands Pascal, but not its types.

If your C program has two structures with elements that have the same name but different types or offsets, **sdb** will be unable to distinguish which element belongs to which structure.

SEE ALSO

adb(1), sh(1sh).

NAME

sdiff — side–by–side difference program

SYNOPSIS

sdiff [**—w** *n*] [**—l**] [**—s**] [**—o** *output*] *filename1 filename2*

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 *filename1*, a > in the gutter if the line only exists in *filename2*, and a ¦ (pipe) for lines that are different. (See EXAMPLES Section.)

OPTIONS

—l Only print the left side of any lines that are identical.

—o *output*
Use the next argument, *output*, as the name of a third file that is created as a user–controlled merging of *filename1* and *filename2*. Identical lines of *filename1* and *filename2* 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.

The editor used will be determined by the value contained in the environment variable **EDIT**. If it has not been set, **ed** is used by default.

—s Do not print identical lines.

—w *n*
Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.

EXAMPLES

Given the file *foo.1* composed of the letters x, a, b, c, and d on individual lines, and the file *foo.2* similiarly composed of the letters y, a, d, and c, the following results will ocurr:

```
sdiff foo.1 foo.2
x                                       | y
a                                         a
b                                       <
c                                       <
d                                         d
                                        > c
```

VARIABLES

EDIT	The editor to be used with the **—o** option.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Sdiff executes *diff(1)* in order to process the input files. If a line coming from **diff** is longer than 200 characters, an error message is printed and **sdiff** exits.

SEE ALSO

diff(1), ed(1), sh(1sh).

NAME

section — print sections of manual entries

SYNOPSIS

man [**—h**] [**—n**] [**—{qv}**] [**—{rt}**] [**—s** *section-list*] [*section*] *title ...*

DESCRIPTION

Like *help(1man),* **section** executes *man(1man)* with the given *section* and *title* arguments in order to obtain the names of the files containing the manual entries. The data generated by *buildif(1man)* must be present in the manual page entry for *section* to be used on that entry.

The *section-list* is a list of abbreviations for sections separated by commas, tabs, and spaces. The following table shows the known abbreviations and the manual page sections they correspond to. The abbreviations are listed in the order that **section** with no **—s** option uses.

na	NAME
sy	SYNOPSIS
de	DESCRIPTION
op	OPTIONS
ex	EXAMPLES
fi	FILES
di	DIAGNOSTICS
va	VARIABLES
rv	RETURN VALUE
ca	CAVEATS
se	SEE ALSO
re	REFERENCES
no	The contents of the notes file in $HOME/.helpnotes

See the manual page for *help(1man)* for a description of manual page notes.

There are three output formats produced by **section** : raw, squeezed, and full. The raw format is used when the **—r** option is given, or when no formatting options are given and the standard output is not a terminal. With raw formatting, the text is printed on the standard output exactly as it appears in the manual page file. The squeezed format is used whenever the **—t** option is given. This format causes all sequences containing backspaces to be squeezed into just the text that would be seen, and all multiple blank lines to be squeezed into a single blank line. When no formatting options are given and the standard output is a terminal, full formatting is used. Full formatting acts like squeezed formatting, except that bold and underlined terminal sequences are printed so that the text looks like it has been piped through *more(1)* or *ul(1).*

Unless the **—h** option is given, the title and section of the manual page is printed on a line before any text. For example, the command "section csh" would print the header "csh(1csh):" before the text.

OPTIONS

—h Do not print header line.

—n Do not print the first line of the text. This first line contains the section header, such as SYNOPSIS.

—q Print only urgent error messages. This option can not be used with **—v.**

—r Raw format. No processing of backspaces or multiple blank lines is done.

—s *section-list*
Use the given section list as the ordering for printing the sections. The list is as described above.

—t Squeeze format. Multiple blank lines are squeezed into single blank lines, and backspace sequences are turned into the characters to be printed.

—v Verbose. Print error messages when a manual entry does not contain the requested section.

EXAMPLES

The following example shows a *csh(1csh)* alias which will print the SYNOPSIS and OPTIONS sections for the given arguments. Note that no header or section name is printed, and only urgent messages are printed.

```
alias ops 'section -qnh -s sy,op \!$'
```

The following is the same example done as a *sh(1sh)* function.

```
ops()
{
section -qnh -s sy,op "$@"
}
```

FILES

$HOME/.helpnotes	Directory in which to search for the notes on the manual entry when the "no" section is requested.

VARIABLES

HOME	The user's home directory.
TERM	The user's terminal type.
TERMCAP	The name of the terminal capability database or the terminal entry itself.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.

CAVEATS

The bold terminal sequence is determined by looking in the termcap entry for "md" and "me". If those do not exist, "so" and "se" are used. If those do not exist, nothing is printed for bold mode.

The underlining terminal sequence is determined by looking in the termcap entry for "uc". If that does not exist, "us" and "ue" are used. If those do not exist, "so" and "se" are used. If those do not exist, nothing is printed for underline mode.

SEE ALSO

apropos(1man), buildif(1man), help(1man), makewhatis(1man), manintro(1man), more(1), ul(1), whatis(1man), man(5man), manindex(5man), termcap(5t), whatis(5man), catman(8man).

NAME

sed — stream editor

SYNOPSIS

sed [**—n**] [**—g**] *scripts* [*filename ...*]

DESCRIPTION

Sed copies the named files (standard input default) to the standard output, edited according to a script of commands.

The *scripts* argument is actually a set of arguments which contain the editing scripts (*script*) and script files (*sfilename). Scripts* can be in one of the following forms :

[**—e**] *script*	– the script is on the command line
—f *sfilename*	– the script is in the file

Any number of script and script *filename* arguments can be given. If there is only one script and it is on the command line, the **—e** is optional.

Normally, all input lines are copied to the standard output. The **—n** option causes only those lines that are explicitly printed by the editing commands to be copied.

The **—g** option causes all occurrences of the substitute command (see the editing command descriptions) to be global, as if each has the **g** flag.

A script consists of comments and editing commands, one per line. Comments are lines beginning with the character '#', and are ignored by **sed.** Editing commands are 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.

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, */regular expression/*. The context address is in the style of *ed(1)* and is modified in the following way:

> The escape sequence **\n** matches a newline embedded in 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 linenumber first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to nonselected pattern spaces by use of the negation function **!** (below).

An argument denoted *text* consists of one or more lines, all but the last of which end with a backslash (\) to hide the newline. 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.

An argument denoted *rfilename* or *wfilename* must terminate the command line and must be preceded by exactly one blank. Each *wfilename* is created before processing begins.

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

(1) **a**
text
Append. Place *text* on the output before reading the next input line.

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

(2) **c**
text
Change. Delete the pattern space. With zero or one address or at the end of a two-address range, place *text* on the output. Start the next cycle.

(2) **d**
Delete the pattern space. Start the next cycle.

(2) **D**
Delete the initial segment of the pattern space through the first newline. Start the next cycle.

(2) **g**
Replace the contents of the pattern space by the contents of the hold space.

(2) **G**
Append the contents of the hold space to the pattern space.

(2) **h**
Replace the contents of the hold space by the contents of the pattern space.

(2) **H**
Append the contents of the pattern space to the hold space.

(1) **i**
text
Insert. Place *text* on the standard output.

(2) **n**
Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

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

(2) **p**
Print. Copy the pattern space to the standard output.

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

(1) **q**
Quit. Branch to the end of the script. Do not start a new cycle.

(2) **r** *rfilename*
Read the contents of *rfilename*. 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 the slash (/). For a fuller description see *ed(1)*. *Flags* is zero or more of the following:

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 *wfilename*
Write. Append the pattern space to *wfilename* if a replacement was made.

(2) **t** *label*
Test. Branch to the colon (:) 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** *wfilename*
Write. Append the pattern space to *wfilename*.

(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 linenumber 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.

OPTIONS

—e *script*
The script is on the command line.

—f *sfilename*
The script is in *sfilename*.

—g Causes all occurrences of the substitute command (see the editing command descriptions) to be global, as if each has the **g** flag.

—n Causes only those lines that are explicitly printed by the editing commands to be copied.

EXAMPLES

The following example copies the contents of the file *orig* to the file *new*, deleting lines beginning with **a** and ending with **e**:

```
sed 'd/^a.*e$/' orig > new
```

This example copies the contents of the file *orig.c* to the file *new.c*, changing all references to system include files (like *#include ⟨stdio.h⟩*) to references to local include files with the same name (\t represents the tab character):

```
sed 's/^\(#[ \t]*include[ \t][ \t]*\)⟨\(.*\)⟩$/\1"\2"/'
  orig.c ⟩ new.c
```

This example copies the contents of the file *orig* to the terminal, deleting all blank lines, and also performing the editing commands in the file *example.sed*:

```
sed -e 'd/^[ \t]*$/' -f example.sed orig
```

The following examples are equivalent. Both copy the contents of the file *orig* to the file *new*, replacing all occurrences of *abc* with *123*:

```
sed '1,$s/abc/123/g' orig ⟩ new
sed -g '1,$s/abc/123/' orig ⟩ new
```

This example copies the file *input* to the standard output, replacing the first occurence of the sequence *the* with *a*, and copying all other lines as is:

```
sed 's/the/a/
t found
b
:found
n
b found' input
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

[P_ERR] A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

The number of distinct files that can be written by the **w** command is limited to the number of files that can be open in a program at the same (subtract one from this if input is not from standard input).

SEE ALSO

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

NAME

send — send a message

SYNOPSIS

send [*filename*][**—draft**][**—help**]
[**—verbose**][**—backup**][**—remove**][**—write**][**—wait**]
[**—nobackup**][**—noremove**][**—nowrite**][**—nowait**]

DESCRIPTION

Send will cause the specified file (default *⟨mh-dir⟩/draft*) to be delivered to each of the addresses in the `To:`, `Cc:`, and `Bcc:` fields of the message. If **—verbose** is specified, **send** will monitor the delivery of local and net mail. **Send** with no argument will query whether the draft is the intended file, whereas **—draft** will suppress this question.

If a `Bcc:` field is encountered, its addresses will be used for delivery, but the `Bcc:` field itself will be deleted from all copies of the outgoing message. Prior to sending the message, the fields `From: user`, and `Date: now` will be prepended to the message.

Once the message has been mailed (or queued) successfully, the file will be renamed with a leading comma, which allows it to be retrieved until the next draft message is sent. The flag **–nobackup** will cause the draft to be removed and no backup will be kept.

If **—nowait** is specified, then **send** will return immediately otherwise it will wait for the mail to be delivered. The mail delivery program (**sendmail**) will print out any error messages directly, but if **—nowait** is specified the error messages will be mailed back. The option **—write** tells the delivery program to write the errors back to the terminal.

Normally, the sender's name is removed from any alias expansions, but if **—noremove** is specified the sender will be included.

If an `Fcc: folder` is encountered, the message will be copied to the specified folder in the format in which it will appear to any receivers of the message. That is, it will have the prepended fields and field reformatting.

Your *.mh_profile* can contain the following entries:

`Path:` To determine the user's MH directory

Send has the following defaults:

filename defaults to draft
—noverbose
—wait
—backup
—nowrite
—remove

Send has no effect on the current message or folder.

FILES

$HOME/.mh_profile The user profile

SEE ALSO

comp(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), dist(1mh), show(1mh), sendmail(8).

NAME

sessions — run UTek Learning Sessions

SYNOPSIS

sessions

DESCRIPTION

The UTek Learning Sessions teach you to use the most basic tools of the UTek operating system. There are six sessions:

Using the Sessions. Shows you how to enter commands and move around in the sessions.

Online Help. Explains the kinds of online help available, and lets you try them.

Files and Directories. Teaches you the fundamental commands for handling files and directories.

Creating a Text File. Teaches you how to create a file using a text editor (**vi**).

Electronic Mail. Helps you set up your electronic mail system and send mail.

Running and Editing a Shell Program. Gives you a sample program. You execute it, change a few lines using an editor, and then execute it again.

CAVEATS

Press the ⟨RETURN⟩ key only when prompted; if you try to "hurry up" the sessions by pressing ⟨RETURN⟩ repeatedly, you will skip screens.

SEE ALSO

manintro(1man), help(1man).

REFERENCES

6130 System Learning Guide.

NAME

set, unset — set and unset shell variables (**csh** built-in)

SYNOPSIS

set [*expression...*]
unset *pattern*

DESCRIPTION

The **set** command is used to set and change the values of *csh(1csh)* variables. Shell variables can either be single words or vectors (arrays) of words. If no arguments are given, the names and values of all shell variables are printed, with vectors surrounded by parentheses. The *expression* arguments may have the following forms:

name — The named variable is set to the null string.

name = *word* — The named variable is set to the word given. The word is subject to substitutions, so special characters must be escaped or quoted.

name [*index*] = *word* — The *index*'th component of the vector *name* is set to the value of *word*. The component must already exist.

name = *(wordlist)* — The named vector is set to the list of values in the *wordlist*.

In all cases, variable expansion is done before any assignment, so the command sequence

```
set x=hello
set x=goodbye y=(you say $x)
```

will set the variable x to "goodbye" and the vector y to "(you say hello)", not "(you say goodbye)".

The **unset** command is used to delete variables that match the given *pattern*. The command **unset** * unsets all variables.

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

Though shell variables are not environment variables (see *setenv(1csh)*), the variables *path,* and *cdpath,* are imported by the shell, and the values of *home, term, user, path,* and *cdpath* are exported to the environment. The shell variable *path* and the environment variable PATH are always maintained together, whereas *cdpath* and CDPATH are maintained together but the values may be changed (see *cd(1sh)* and *cd(1csh)* for

more information).

Following is a description of variables which affect built-in commands:

argv Set to the arguments to the shell, it is from this variable that positional parameters are substituted; for example, $1 is replaced by **$***argv[1]*, and so forth.

cdpath Gives a list of alternate directories searched to find subdirectories in **chdir** commands. This variable is maintained along with the CDPATH environment variable.

complete When set, this variable enables command completion in interactive shells.

cwd The full pathname of the current directory. See CAVEATS.

dirname Set to the default directory to change to when the argument to *cd(1csh)* or *pushd(1csh)* is not found in the current directory or in the list of directories found in the *cdpath* variable.

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. In all cases, redirection is not taken into account, so the commands

```
set echo
cat /etc/group )& foo
```

will result in echoing "cat /etc/group" to the original standard error, instead of into the file "foo".

hardpaths When symbolic links are followed when changing directories, the value of the variable *cwd* and the values on the directory stack may not be proper pathnames. When this variable is set, each change of directory causes the current directory to be expanded to not include symbolic links. This should only be used when accuracy is required, as it makes directory changes much slower.

histchars Can be given a string value to change the characters used in history substitution. The first character of its value is used as the history substitution character, replacing the default character !. The second character of its value replaces the character ↑ in quick substitutions.

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 the tilde (~) 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 ⟨CTRL–D⟩'s.

list When set to nothing, this variable enables file and command listing using the ^D character in interactive mode. If set to 'f', filename marking is turned on. If set to 'l', filename marking is turned on and symbolic links are specially marked.

listpathnum This modifies the output of file listing (see the *list* variable) so that the number of the path element where each of the commands is found is printed.

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 filename* when there is mail in *filename.*

If this variable is not set, no checking for mail is done.

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; for example, **echo [** still gives an error.

notify If set, the shell notifies asynchronously of job

completions. The default is to gather present job completions just before printing a prompt.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable then only full pathnames will execute. The usual search path is ., */bin*, and */usr/bin*, but this may vary from system to system. For the superuser, 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 superuser.

savehist is given a numeric value to control the number of entries of the history list that are saved in ~*/.history* when the user logs out. Any command which has been referenced in this many events will be saved. During start up, the shell sources ~/.history into the history list enabling history to be saved across logins. Too large values of **savehist** will slow down the shell during start up.

shell The file in which the shell resides.

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

time Controls automatic timing of commands and time summary format. If the first value in the vector is set, then any command which takes more than this many CPU seconds will cause a time and resource usage summary to be printed when it terminates. If the second value in the vector is set, that value controls the summary format. See *time(1csh)* for the summary format values and the default format.

vbell If set to nothing, command completion errors will be signaled by the visible bell (:vb:) sequence from the termcap entry (see *termcap(5t))* instead of the terminal bell. If set to anything else, the value of the variable is printed.

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

EXAMPLES

The following command line sets the value of the variable 'Here' to the current working directory, the vector 'files' to the names of the files in the current directory which begin with the letter 't', and the fourth element of the variable 'junk' to the word "foo". If 'junk' does not already exist and have at least four elements, an error message will result.

```
set Here="`pwd`" files=( t* ) junk[4]=foo
```

This example unsets the values of all single-letter variables.

```
unset ?
```

DIAGNOSTICS

name: undefined variable.
An element of the variable *name* was assigned to, but the name has never been assigned.

set: subscript out of range.
An element of a variable was assigned to, but that element does not exist.

CAVEATS

The value of the variable *cwd* is not always correct, especially when symbolic links are followed. If this variable is required to be correct, the variable *hardpaths* should be set.

When command completion or file listing are turned on, terminal echo mode will always be turned on and the terminal will always be set to cooked input mode. This is done to ensure that command completion and file listing behave properly. If you need to change these terminal characteristics, you must turn off command completion and file listing.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1sh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1sh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

set — set shell options and parameters (*sh* built–in)

SYNOPSIS

set — [—, **a**, **e**, **f**, **h**, **k**, **n**, **t**, **u**, **v**, **x**][*arg* . . .]

DESCRIPTION

Set is used to set options and reset positional parameters. If the first argument begins with a dash (–) or a plus sign (+), it is interpreted as options to the shell. A leading dash (–) turns the option on, and a plus sign (+) turns the option off.

All remaining arguments are positional parameters and are assigned, in order, to $1, $2, and so forth. The lone dash (–) is not assigned.

If no arguments are given, the values of all shell variables, except for those set internally, are printed.

The options for **set** can also be used upon invocation of the shell. The current set of options may be found in *$-*.

OPTIONS

——

Null option. Useful for setting positional parameters when the first parameter begins with a dash (–).

The following options are shown with a leading dash (–), which can be replaced by a plus sign (+) for the reverse effect:

—a Mark variables which are modified or created for export to the environment.

—e If noninteractive, exit immediatly if a command fails.

—f Disable filename generation.

—h Locate and remember commands in functions as functions are defined. Normally, function commands are located when the function is executed.

—k All keyword parameters 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.

EXAMPLES

The following command sets the **—x** and **—e** options in the current shell:

```
set -xe
```

The following shell script prints the name and size of all files in the current directory not owned by the user specified on the command line. The default user is the user that invokes the script.

```
#!/bin/sh
Userid=${1-`whoami`}
for i in .* *
{
        set - `ll -d "$i"`
        if test "$Userid" != "$3"
        then
                echo "$i\t\t$4"
        fi
}
```

RETURN VALUE

[NO_ERRS] Command completed without error.

[USAGE] Incorrect command line syntax. Execution terminated.

CAVEATS

It is not possible to set and unset options in the same invocation of **set**.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), getopt(1), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2).

NAME

setenv, unsetenv — change environment variables (**csh** built–in)

SYNOPSIS

setenv *name value*
unsetenv *pattern*

DESCRIPTION

Environment variables, unlike shell variables, are passed to all child processes in the "environment" (see *environ(7)).*

The command **setenv** sets the environment variable *name* to the given *value,* which must be a single word.

The command **unsetenv** removes all environment variables whose name matches *pattern,* which may contain any metacharacters normally used in filename expansion. This deletes the names from the environment for all child processes, leaving the parent processes unaffected.

Setting the variable PATH results in the shell variable *path* being set similarly. See *set(1csh)* for more information.

EXAMPLES

The following command sets the environment variable PATH to the value ':/bin:/etc:/usr/bin'. Note that this causes the shell variable *path* to be set to '(. /bin /etc /usr/bin)'.

```
setenv PATH :/bin:/etc:/usr/bin
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the message occurred.

CAVEATS

Shell variables take precedence over environment variables. This means that setting the environment variable 'prompt' to 'foo' does not change the prompt for the current or subsequent shells, and that the value of '$prompt' is not 'foo' in a subsequent invocation of **csh.**

There is no way for a child to change the environment of the parent.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), export(1sh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), printenv(1), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), set(1sh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), getenv(3c), environ(7).

NAME

sh — command language

SYNOPSIS

sh [**—acefhiknrstuvx**] [*arg*] ...

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. This document describes the syntax of the language understood by **sh.**

CONTROL STATEMENTS:

for *name* [**in** *word* ...] **do** *list* **done**
for *name* [**in** *word* ...] { *list* }

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

while *list* [**do** *list*] **done**

until *list* [**do** *list*] **done**

(*list* **)**
Execute *list* in a subshell.

{ *list*; }
List is simply executed.

name **()** { *list*; }
Define a function which is referenced by *name.* The body of the function is a list of commands between { and }. See the section *Functions* for more information.

. *filename*
Commands from the named file are read and executed by the current shell. The search path specified by the environment variable **PATH** is used to find the file.

: ...
The null command. All arguments are evaluated. This can be used as a comment.

...
The comment character. Ignore all text until the end of the line.

command ¦ *command*
Both commands are executed, with the standard output of the first *command* connected to the standard input of the second *command.* This is called a "pipe".

command **&&** *command*
The first *command* is executed, and if the exit status returned is nonzero, the second *command* is executed.

command || *command*
The first *command* is executed, and if the exit status returned is zero, the second *command* is executed.

‘ *command* ‘
Is replaced by the output from *command*. Backquotes may be nested by escaping the ‘ character using a backslash character (\). The number of backslashes required for each level of nesting is twice that of the previous level plus one. So, for example, the first level of nesting requires one backslash, the second requires three, the third seven, and so on. There is no imposed limit to the number of nesting levels.

In the above, *name* refers to a shell variable name (without the preceding $), *word* refers to an entity which expands to list of one or more sequences of characters separated by the field separator (see the **IFS** variable), *list* refers to one or more commands separated by newlines or semicolons, *pattern* refers to a single sequence of characters possibly containing the special characters *, [,], and ? (see *FILENAME GENERATION*), and *command* refers to an executable file or builtin command name, followed by zero or more command arguments.

INTERACTIVE SHELLS

When an interactive shell is started and the variable **ENV** is set to the name of an existing file, commands are read from that file and executed. This is usually used to define functions. For security reasons, this only happens for interactive shells. There is no default value for **ENV**, so this file must be explicitly executed from, if desired, in the *.profile* file (described below).

If the name of the shell begins with a dash (-), commands are read from the files */etc/profile* (or */etc/rprofile* for restricted shells) and *.profile* (note that the *.profile* file in the current directory is executed, and that most programs that start up shells with the intent of executing the user's *.profile* should change directory to the user's home directory). Shells with names beginning with a dash (-) are executed by *login(1)* and possibly by *su(1)*.

PARAMETER SUBSTITUTION

The character **$** is used to introduce substitutable parameters (also called variables). Variables may be set by writing

name=value [*name=value*] ...

$ {*parameter*}
A *parameter* is a sequence of letters, digits or underscores (a *name*), a digit, or any of the characters * **@ # ? — $!**. 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. Variable names containing only digits (such as $1) are called *positional parameters*. These are set to the arguments given to the current shell script.

Pattern matching is not performed on the *value* given.

The following are special parameter substitution forms. In each of the forms, the ':' following the *parameter* is optional. If given, the *parameter* must be both set and non–null for it to be substituted, as opposed to only having to be set when the ':' is not given.

$\{*parameter*:—*word*\}
If *parameter* is set, substitute its value; otherwise substitute *word.*

$\{*parameter*:= *word*\}
If *parameter* is not set, set it to *word;* the value of the parameter is then substituted. Positional parameters may not be assigned in this way.

$\{*parameter*:?*word*\}
If *parameter* is set, substitute its value; otherwise, print *word* and exit from the shell.

$\{*parameter*:+ *word*\}
If *parameter* is set, substitute *word;* otherwise substitute nothing.

For a description of parameters set and used by the shell, see the VARIABLES section.

FILENAME GENERATION

If one of these characters appears, the word is regarded as a pattern. Except when used as the pattern in a *case* statement, the word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The dot character (**.**) at the start of a filename or immediately following a slash (/) and the slash character (/) must be matched explicitly.

***** Matches any string, including the null string.
? Matches any single character.
[...]
Matches any one of the characters enclosed. A pair of characters separated by a dash (—) matches any character lexically between the pair.

The option **—f** turns off filename generation.

QUOTING

Characters between single quotes are not processed. Characters between double quotes have macros and commands substituted, but **IFS** characters are ignored (see VARIABLES).

"$*" is equivalent to **"$1 $2 ..."** whereas
"$@" is equivalent to **"$1" "$2"** ... .

INPUT OUTPUT

⟨filename
Use *filename* as standard input.

›*filename*
Use *filename* as standard output. If the file does not exist, it is created; otherwise it is truncated to zero length.

››*filename*
Use *filename* as standard output. If the file exists, output is appended; otherwise the file is created.

‹‹**[**—**]***string*
The shell input is read up to a line the same as *string*, or end–of–file. The resulting document becomes the standard input. If any character of *string* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, \‹*newline*› is ignored, and a backslash (\ ) is used to quote the characters \ , $, ' , and the first character of *string*. If a dash (–) is appended to ‹‹, all leading tabs are stripped from *string* and from the document.

‹**&***digit*
The standard input is duplicated from file descriptor *digit;* 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, 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 to be a duplicate of file descriptor 1. The standard for file descriptor numbers is that 0 is *standard input,* 1 is *standard output,* and 2 is *standard error.*

FUNCTIONS

If a given command name does not match a builtin command but matches a defined function, the commands associated with the function are executed in the current shell (similar to the **alias** mechanism in *csh(1csh)).* The positional parameters **$1, $2, ...** are set to the arguments of the function. A function is executed just like any other program with one major exception: if the function is executed in the current shell, its arguments replace the current positional parameters, so the original parameter values are lost. This can be prevented by running the function in a subshell.

Note that functions with names the same as builtin commands are ignored.

The command *lsh(1sh)* is a limited version of **sh**. In some systems, this is called **rsh**, but is called **lsh** here because of the remote shell command, **rsh**.

OPTIONS

—a Mark all variables which are modified or created for export (see *export(1sh)).*

—c *string*
If the **—c** flag is present, commands are read from *string*.

—e If noninteractive, exit immediately if a command fails.

—f Disable filename generation.

—h Hash commands used in functions when functions are defined.

—i If the **—i** flag is present or if the shell input and output are attached to a terminal (as told by **gtty**) then this shell is *interactive*. In this case the terminate signal SIGTERM (see *signal(3c)*) is ignored (so that the input **kill 0** does not kill an interactive shell) and the interrupt signal SIGINT is caught and ignored (so that **wait** is interruptible).

—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.

—s If the **—s** flag is present or if no arguments remain, then commands are read from the standard input. Shell output is written to file descriptor 2.

—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.

FILES

$HOME/.profile	This file is read and commands contained in it executed when the shell is called as *—sh*, usually at login.
*/tmp/sh**	Temporary storage for storing arguments to ⟨⟨.

DIAGNOSTICS

Errors detected by the shell, such as syntax errors cause the shell to return a nonzero 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 **exit**).

VARIABLES

These variables are set by the shell:

$0	The name of the current shell or script.
$1, $2, ...	Arguments given to the current shell or script.
$*	$1, $2, ... collectively.
$@	Same as $*, except when quoted.
$#	The number of positional parameters.

$-	Options currently set.
$?	The exit status of the last command executed.
$$	The process ID of this shell.
$!	The process ID of the last background command invoked.

These variables are used but not set by the shell.

$CDPATH	The search path for the **cd** command.
$HOME	The user's home directory.
$PATH	The search path for command execution.
$MAIL	The name of the file to be searched for notification of new mail.
$MAILCHECK	The number of seconds between checks for new mail (default is 600).
$MAILPATH	Colon–separated list of files to check for mail new mail.
$PS1	Primary prompt string (default $).
$PS2	Secondary prompt string (default >).
$IFS	Field separator (default tab, space, and newline).
$SHELL	The name of the shell. If the name has an **r** in it, the shell is restricted.
$ENV	The name of the startup file for all interactive shells. Used for defining functions.

CAVEATS

Builtin commands such as **set** and **read** may not have their input or output redirected. In order to redirect a command like **set,** it must be executed in a subshell or by using **exec**. For example, the command

```
(set) > set.out
```

will print the values of *exported* names. Note that this does not produce the same output as would **set** in the current shell, since not all variables are exported.

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 the file by another name.

Signal 11 (segmentation violation) can not be trapped.

The execution path is hashed for faster execution. See the manual page for *hash(1sh)* for more information.

The first line of all shell scripts should be of the form *#!/bin/sh* or *#!/bin/csh* (see *execve(2)* for more information). In some systems, executable files whose first line is not of the form described above and whose first line begins with #, are executed by *csh(1csh)*. This is not true in this system.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), lsh(1sh), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2).

NAME

shift — reposition vector variables (**csh** built–in)

SYNOPSIS

shift [*variable*]

DESCRIPTION

The **shift** command shifts all memebers of the named *variable,* or *argv* if none given, discarding element number 1.

EXAMPLES

The major use of **shift** is in parsing arguments. The following shell script fragment parses the arguments given to it, allowing the arguments **—a, —b,** and **—c,** and printing an error message for others. At the end of the parsing, *argv* contains the remaining arguments.

```
#!/bin/csh -f
set aflag="0" bflag="0" cflag="0"
while ($#argv)
        if ("x"$argv[1] =~ x-*) then
                switch ($argv[1])

                case -a:
                        set aflag=1
                        breaksw

                case -b:
                        set bflag=1
                        breaksw

                case -c:
                        set cflag=1
                        breaksw

                default:
                        echo "$0 : Unknown option $argv[1]"
                        exit 1

                endsw
                shift
        else
                break
        endif
end
```

RETURN VALUE

[NO_ERRS] Command completed without error.

[1] An error of the type described in the message occurred.

CAVEATS

This version of **shift** is different from *shift(1sh)* in that it does not take an argument for the number of shifts to do at once.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1sh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

shift — rename positional parameters (**sh** built-in)

SYNOPSIS

shift [*n*]

DESCRIPTION

The positional parameters from **$***n* + 1 . . . are renamed to **$1** . . . . If *n* is not given, it is assumed to be 1.

EXAMPLES

The major use of **shift** is to parse command lines. This shell script checks its argument to see if it is a number. If it is, it may be doubled, tripled, or quadrupled by giving the options **—d, —t,** or **—q,** respectively, to the shell script. Only the last flag is used. In this case, **shift** is used so that the argument being worked with is always $1.

```
#!/bin/sh
Mult=1
while true
do
        case $1 in

                -d)
                        Mult=2
                        ;;

                -t)
                        Mult=3
                        ;;

                -q)
                        Mult=4
                        ;;

                -*)
                        echo "$0 : usage : "`basename $0`" [-d] [-t] [-q] num"
                        exit 1
                        ;;
                *)
                        break
                        ;;
        esac
        shift
done
if test $# -ne 1
then
        echo "$0 : usage : "`basename $0`" [-d] [-t] [-q] num"
        exit 1
```

```
fi
if test "0$1" -eq 0 -a "0$1" != "00"
then
        echo "$1 is not a positive integer."
        exit 1
fi
expr "$1" '*' "$Mult"
```

RETURN VALUE

[NO_ERRS] Command completed without error.

[NP_ERR] An error occurred that was not a system error. Execution terminated.

CAVEATS

It should be noted that only positional parameters $1 through $9 may be referenced individually. To get the value of $12, **shift** must be executed three times or with a value of 3, at which point the desired value will be in $9.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1csh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2).

NAME

show — show (list) mail messages

SYNOPSIS

show [*+folder* **] [msgs] [-draft] [-header]**
[-noheader] [-format] [-noformat] [-pr]
[-nopr] [-help] [switches for **pr]**

DESCRIPTION

Show lists each of the specified mail messages to the standard output (usually your terminal). The messages are listed exactly as they are, with no reformatting.

If you specify the **—pr** option, then *pr(1)* is invoked to list the messages and the switches (other than those recognized by **show**) are passed to **pr**.

Your *.mh_profile* file can contain the following entries:

`Path:` To determine the user's MH directory
`Current-Folder:` To find the default current folder
`Show:` To set options of **show**

Show has the following defaults:

—format
—header
—nopr
msgs defaults to current message
+folder defaults to current folder

If you specify a folder with the *+folder* option, it becomes the current folder. The last message listed becomes the current message.

You can use the keywords **all**, **cur**, **next**, and **prev** for the *msgs* argument. These keywords have the following meanings:

all Display all the messages in a folder.

cur Display the current message in a folder.

next Display the next message (the one after the current message) in a folder.

prev Display the previous message (the one before the current message) in a folder.

OPTIONS

—draft
Print the draft file, *mh-directory/draft*, if it exists.

—format
Use the default pagination program to display the specified mail messages (same as **—nopr**). The **—pr** and **—noformat** options suppress this option.

—noformat
Use *cat(1)* to display the specified mail messages. The **—pr** option suppresses this option.

—header
Print a header telling which message is being displayed. The header is in the following format:

(Message *folder*:*number*).

—noheader
Don't print the header described for the **—header** option.

—help
Print a usage message for **show**.

—pr
Invoke **pr** as the pagination program. The **—pr** option overrides the **—format** and **—noformat** options.

—nopr
Invoke the default pagination program.

FILES

$HOME/.mh_profile	The user profile
/bin/pr	Pagination program, *pr(1)*

SEE ALSO

comp(1mh), folder(1mh), forw(1mh), inc(1mh), mail(1mh), next(1mh), pick(1mh), prev(1mh), prompter(1mh), refile(1mh), repl(1mh), rmf(1mh), rmm(1mh), scan(1mh), send(1mh), show(1mh).

NAME

siline — generate a line given slope and intercept

SYNOPSIS

siline [—c*n*] [—i*n*] [—n*n*] [—s*n*] [*vector* ...]

DESCRIPTION

Output is a vector of values *slope*x+intercept*, where *x* takes on values from *vector(s)*. If the **-n** option is given, *vector* is the ascending positive integers. If neither the *n* option nor a *vector* is given, *vector* comes from the standard input.

OPTIONS

—c*n*
n is the number of output elements per line.

—i*n*
n is the *intercept*, 0 if not given.

—n*n*
n is the number of positive integers to be used for *x*.

—s*n*
n is the *slope*, 1 if not given.

EXAMPLES

The following example outputs a simple linear fit of vector B on vector A. (The o option of *lreg(1g)* outputs the slope and intercept in option form of B regressed on A.)

```
siline -`lreg -o,FA B` A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

sin — sin

SYNOPSIS

sin [**—c***n*] [*vector ...*]

DESCRIPTION

Output is the sin for each element of the input *vector(s)*. Input is assumed to be in radians. If no *vector* is given, the standard input is assumed.

OPTIONS

—c*n*

n is the number of output elements per line.

EXAMPLES

The following example outputs the sin of each element of A, three per line.

```
sin -c3 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

size — size of an object file

SYNOPSIS

size [*object . . . filename*]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, and **bss** portions, and their sum in hex and decimal, of each *object . . . filename* argument. If no file is specified, *a.out* is used.

EXAMPLES

The following returns the size of the file *a.out* in the local directory:

```
size
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

a.out(5).

NAME

sleep — suspend execution for an interval

SYNOPSIS

sleep *time*

DESCRIPTION

Sleep suspends execution for *time* seconds.

EXAMPLES

It can be 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
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

Time (in seconds) must be less than the maximum number allowed in an unsigned long.

SEE ALSO

alarm(3c), sleep(3c).

NAME

smake — update files from another machine

SYNOPSIS

smake [**—f** *smakefilename*] [**—h**] [**—i**] [**—n**] [**—r**] [**—v**] [**—X** *re*] [**—Y** *re*] [*sources*] [*destination*]

DESCRIPTION

Smake is used to update files on the current machine from files on other machines. The syntax and function is similar to **rcp**, except that **smake** will only copy a file if the target is older than the source. **Smake** also makes the target the same mode, user, group, and modification date as the source. Its advantages over **rcp** include setting the modes, only doing necessary copying, and being usable by root.

The destination file must be on the host machine. Sources may be either on the host machine, or, if the name is preceded by a host name followed by a colon, on another machine.

Normal **csh** special characters (~, {, *, ?, []) will be interpreted as expected. **Smake** expands the arguments given it either locally or remotely. Note that if it is desired to match a filename with a glob character in it, the glob character must be escaped. If the filename was given on the argument line, it will have to be escaped twice, since **smake** will expand it once more after the shell expands it.

An additional character, # , is used to select the highest version of something. It matches the highest version number, where a version is a string from the set {0..9, .(dot)}, and the ordering (higher) is on the numeric fields within that string.

Smake will also read a list of arguments from a file specified with the **—f** flag. Each line in the file is treated as if it were appended to the string **smake**, and executed by **csh**. This is slightly more efficient than specifying each line separately, since **smake** will only initialize a connection to a host once. However, it allows commands through **make**.

Each line in the **smake** file may be followed by a sequence of command lines. The command lines must start with at least one tab, and if present, will be the only action taken by **smake**, instead of updating the files. Source files and target files are compared as before, but each pair that requires updating causes an execution of the command line. The command line is sent to the **csh**, with the variable $*T* set to the name of the target, and the variable $*S* set to the name of the source.

A target is a file that is going to be updated on the local machine; it has a corresponding source. Directories are not considered targets, and do not cause an invocation of the command line.

OPTIONS

The following arguments are accepted by **smake**:

—f The following argument is a *smakefilename*; each line in the file is broken up into arguments and run as if it had been given directly to **smake** (see above).

—h Normally, a file about to be replaced is renamed to a temporary file, the source is retrieved, and the old file is not deleted until the new one is successfully in place. The **—h** option is used to preserve any hard links to a destination file, which would normally be broken. **Smake** will truncate the old file, and copy the new file into it. This happens only if both source and destination are regular files, and the destination has a link–count greater than 1.

—i The user is asked before each file is copied.

—n No files are actually copied; instead, shell commands to make the necessary changes are printed on *stdout*.

—r Each source file that is a directory is copied recursively to the destination; the destination must be a directory.

—v Verbose. Produce lots of output.

—X The following argument is a regular expression. Any file or directory that matches the **RE** is NOT copied. Note that if a directory name matches the **RE**, nothing in the directory is copied.

—Y The following argument is a regular expression. Only files that match the RE are copied. This test is not applied to directories. This flag may be used at the same time as the **–X** flag.

EXAMPLE

The following *smakefilename* would be read by the command

smake —f ./smakefilename

and update all the files in */bin* and */usr/bin*, and the entire tree */usr/local*:

hammer:*/bin/* /bin*
hammer:*/usr/bin/* /usr/bin*
–r hammer:*/usr/local /usr*

FILES

/etc/smake-dirs	A list of all legal absolute paths prefixes. If SIGHUP is sent to the **smaked**, */etc/smake-dirs* will be re–read.

CAVEATS

Smake is an incredible security hole.

The # character doesn't work yet.

The **—Y** and **—X** options don't work as well as hoped. Remember to escape special characters, and don't even try to use them from the command line. **Smake** doesn't know what the user's shell variables are on the remote hosts. Therefore, it doesn't expand them anywhere except on command lines, where it is done by a shell anyway.

SEE ALSO

regex(3c).

NAME

soelim — eliminate **.so**'s from **nroff** input

SYNOPSIS

soelim [*filename...*]

DESCRIPTION

Soelim reads the specified files or the standard input and performs the textual inclusion implied by the **nroff** directives of the form

```
.so somefilename
```

when they appear at the beginning of input lines. This is useful since programs such as **tbl** do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of a dash (—) is taken to be a filename corresponding to the standard input.

Note that inclusion can be suppressed by using a single quotation mark (') instead of a dot (.).

A sample usage of **soelim** would be

```
soelim exum?.n ¦ tbl ¦ nroff -ms ¦ col ¦ lpr
```

EXAMPLES

Here, inclusion is suppressed in the file */usr/lib/tmac.s*:

```
'so /usr/lib/tmac.s
```

CAVEATS

The format of the source commands must involve no strangeness — exactly one blank must precede and no blanks follow the filename.

SEE ALSO

nroff(1), more(1).

NAME

sort — sort or merge files

SYNOPSIS

sort [**—bcdfimnrut***x*][*+pos1* [*—pos2*]] ... [**—o** *name*]
[**—T** *directory*] [*name*]...

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name — (dash) 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 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 the number of fields to skip from the beginning of the line and *n* tells the 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 nonblank 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 nonempty, nonblank 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.

OPTIONS

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.

—c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort. Only one file may be checked at a time.

—d Dictionary order: only letters, digits, and blanks are significant in comparisons.

—f Fold uppercase letters onto lowercase.

—i Ignore characters outside the ASCII range 040–0176 in nonnumeric comparisons.

—m
Merge only; the input files are already sorted.

—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.**

—**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.

—**r** Reverse the sense of comparisons.

—**t***x*
The tab character separating fields is *x*.

—**u** Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

—**T** The next argument is the name of a directory in which temporary files should be made.

EXAMPLES

Print in alphabetical order all the unique spellings in a list of words. Capitalized words differ from uncapitalized.

```
sort -u +0f +0 list
```

Print the password file (*passwd(5)*) sorted by user ID number (the third colon–separated field).

```
sort -t: +2n /etc/passwd
```

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.

```
sort -um +0 -1 dates
```

FILES

*/usr/tmp/stm** First and second tries for temporary files.

DIAGNOSTICS

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option —**c** are provided.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Lines over 1024 characters in length may be silently truncated.

No more than nine keys may be specified for sorting.

SEE ALSO

awk(1), comm(1), cut(1), egrep(1), fgrep(1), grep(1), join(1), look(1), paste(1), rev(1), uniq(1).

NAME

source — execute commands from file (**csh** built–in)

SYNOPSIS

source [**—h**] *filename*

DESCRIPTION

The **source** command reads commands from *filename* and executes them (unless **—h** is given) in the current shell. **Source** commands may be nested, but if they are nested too deeply, the shell will run out of file descriptors. An error in a **source** at any level terminates all nested **source** commands. Input to **source** commands is not added to the history list when executed.

OPTIONS

—h Add the commands to the history list instead of executing them.

EXAMPLES

The following command line reads commands from the file 'mycmds' and places them in the history list of the current shell.

```
source -h mycmds
```

RETURN VALUE

The return value is the value returned by the last command executed, or 1 if the file does not exist.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

spell — find spelling errors

SYNOPSIS

spell [**—v**] [**—b**] [**—x**] [*filename* . . .]

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, 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 *tbl(1), neqn*(1) constructions.

Under the **—v** option, all words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

Under the **—b** option, British spelling is checked. Besides preferring *centre, colour, speciality, travelled,* and so forth, this option insists upon -*ise* in words like *standardise*, with Fowler and the OED to the contrary notwithstanding.

Under the **—x** option, every plausible stem is printed with an equal sign (=) for each word.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

The stop list filters out misspellings (for example, thier = thy—y + ier) that would otherwise pass.

OPTIONS

—b British spelling is checked.

—v All words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

—x Every plausible stem is printed with an equal sign (=) for each word.

EXAMPLES

The following example will print any words in the file report not found (or derivable from) the spelling list:

```
spell report
```

FILES

/usr/dict/hlista	Hashed spelling, American
/usr/dict/hlistb	Hashed spelling, British
/usr/dict/hstop	Hashed stop list
/usr/dict/words	Unhashed spelling list
/usr/lib/spell	Hashed list search program

RETURN VALUE

[NO_ERRS] Command completed without error.

[P_ERR] A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions. British spelling was done by an American.

SEE ALSO

spellin(1), spellout(1), deroff(1), sort(1), tee(1), sed(1), tbl(1).

NAME

spellin — add words to spelling list

SYNOPSIS

spellin [*list*]

DESCRIPTION

Spellin adds the words on the standard input (one per line) to the preexisting *list* and places a new list on the standard output. If no *list* is specified, the new list is created from scratch.

EXAMPLES

Assuming *new_words* contains a list of words one per line, the following example will add these words to the spelling list in the file */usr/dict/hlista* (provided the user has write permission on */usr/dict/hlista*).

```
cp /usr/dict/hlista /tmp/hlista
spellin /tmp/hlista <new_words >/usr/dict/hlista
```

FILES

/usr/dict/hlista	Hashed spelling list, American
/usr/dict/hlistb	Hashed spelling list, British
/usr/dict/hstop	Hashed stop list

RETURN VALUE

[NO_ERRS]	Command completed without error.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

spell(1), spellout(1).

NAME

spellout — print words missing from spell list

SYNOPSIS

spellout [**—d**] *list*

DESCRIPTION

Spellout looks up each word in the standard input (one word per line) and prints on the standard output those that are missing from (or present on, with option **—d**) the hash list.

OPTIONS

—d Prints all the words that are present in the hash list.

EXAMPLES

Assuming that *checkwords* is a list of words one per line, the following command will print any of the words in *checkwords* that are not in */usr/dict/hlista.*

```
spellout /usr/dict/hlista <checkwords
```

FILES

/usr/dict/hlista	American hashed spelling list
/usr/dict/hlistb	British hashed spelling list
/usr/dict/hstop	Hashed stop list

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

spell(1), spellin(1).

NAME

split — split a file into pieces

SYNOPSIS

split [—*n*] [*filename* [*name*]]

DESCRIPTION

Split reads *filename* and writes it in *n*–line pieces (default 1000), as many as necessary, onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically. If no output name is given, **x** is default.

If no input file is given, or if — (a dash) is given instead, then the standard input file is used.

OPTIONS

—*n* The file is split into *n*–line pieces. The default size is 1000 lines.

EXAMPLES

The following example will split the file *text* into 500 line pieces starting with filename *xaa*:

```
split -500 text
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

The maximum number of files that may be created with the *aa - xx* appendix is 26 X 26 = 676. An error message is printed if an attempt is made to create more than this.

SEE ALSO

csplit(1).

NAME

stop, suspend — stop execution of jobs (**csh** built-in)

SYNOPSIS

stop %*job...*
suspend

DESCRIPTION

The command **stop** sends the stop signal to the named jobs. The resulting job(s) show up on the *jobs(1csh)* listing with the message "Stopped (signal)". The job can be restarted using the commands described in *fg(1csh)*.

The *suspend* command is used to suspend the current shell. This is required since the shell always ignores the stop signal. This is useful with shells started with *su(1)* or other subshells.

EXAMPLES

Assume that the command "make myprog" is job number 1, and is the only *make(1)* job running. The following shows two different ways to stop the job.

```
stop %1
stop %make
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the error message has occurred.

CAVEATS

The **suspend** command is not present in *sh(1sh)*. This is no problem, since the Bourne shell does not ignore the stop signal.

There is no way to suspend a login shell.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), make(1), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), kill(2), killpg(2), signal(3c).

NAME

strings — find the printable strings in a file

SYNOPSIS

strings [**—a**] [**—d**] [**—o**] [**—x**] [*—number*] *filename...*

DESCRIPTION

Strings looks for ASCII strings in a binary file. A string is any sequence of four or more printing characters ending with a newline or a null.

Strings is useful for identifying random object files and many other things.

If the **—d, —o,** or **—x** options are given, the strings are preceded by offsets in decimal, octal, and hex. The order of the offsets (when combinations of these options are given) is: octal, decimal, hex.

OPTIONS

—a Search the entire file. Otherwise, **strings** starts looking at the location where an object file's header would end.

—d Each string is preceded by its offset in the file in decimal.

—o Each string is preceded by its offset in the file in octal.

—x Each string is preceded by its offset in the file in hex.

—number
Number is used as the minimum string length rather than four.

EXAMPLES

The following invocation of this command will print all ASCII strings in the object file *cmd.o*:

```
strings cmd.o
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Without the **—a** option, **strings** assumes it is searching an object file and skips beyond the header text before beginning the search.

SEE ALSO

od(1).

NAME

strip — remove symbols and relocation bits

SYNOPSIS

strip [—**O** *suffix*] [—**S** *suffix*] [—**o**] [—**s**] [—**v**] *file* ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space after a program has been debugged.

The effect of **strip** is the same as when the —**s** option of **ld** is used.

OPTIONS

—**O** *suffix*
Create an output file named *filesuffix* that is a stripped version of *file*. The input file is left unstripped.

—**S** *suffix*
Save stripped information. The symbol table and other information that **strip** removes from *file* are stored in *filesuffix*. The symbol table file can be recombined with *file* using **unstrip**.

—**o** Create an output file named *file.stripped* that is a stripped version of *file*. The input file is left unstripped.

—**s** Save stripped information. The symbol table and other information that **strip** removes from *file* are stored in *file.symtab*. The symbol table file can be recombined with *file* using **unstrip**.

—**v** Verbose. Reports the names of all input, output, and symbol table files used. Also reports sizes (in bytes) of the input and output files.

EXAMPLES

The following invocation removes the symbol table and relocation bits from the file *cmd* and prints messages telling you what it did:

```
strip -v cmd
```

The following invocation creates a stripped version of the file *cmd* in a file named *cmd.out*. The symbol table information is saved in *cmd.symtab*.

```
strip -O ".out" -s cmd
```

FILES

/tmp/stm?	Temporary file

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.

[P_WARN] A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

[P_ERR] A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

ld(1), unstrip(1), a.out(5).

NAME

struct — structure FORTRAN programs

SYNOPSIS

struct [**—a**][**—b**][**—c***n*][**—e***n*][**—i**][**—n**][**—s**][**—t***n*] *filename*

DESCRIPTION

Struct translates the FORTRAN program specified by *filename* (standard input default) into a Ratfor program. Wherever possible, Ratfor control constructs replace the original FORTRAN. Statement numbers appear only where still necessary. Cosmetic changes are made, including changing Hollerith strings into quoted strings and relational operators into symbols (for example, **.GT.** into >). The output is appropriately indented.

OPTIONS

—a Turn sequences of **else-if**'s into a non-Ratfor switch of the form

```
switch
        {       case pred1: code
                case pred2: code
                case pred3: code
                default: code
        }
```

The case predicates are tested in order; the code appropriate to only one case is executed. This generalized form of switch statement does not occur in Ratfor.

—b Generate **goto**'s instead of multilevel break statements.

—c*n* Increment successive labels in the output program by the nonzero integer *n* (default 1).

—e*n* If *n* is 0 (default), place code within a loop only if it can lead to an iteration of the loop. If *n* is nonzero, admit a small code segments to a loop if otherwise the loop would have exits to several places including the segment, and the segment can be reached only from the loop. *Small* is near, but not equal to, the number of statements in the code segment. Values of *n* under 10 are suggested.

—i Do not turn computed **goto** statements into switches. (Ratfor does not turn switches back into computed **goto** statements.)

—n Generate **goto**'s instead of multilevel **next** statements.

—s Input is accepted in standard format; for example, comments are specified by a **c**, **C**, or * in column one, and continuation lines are specified by a nonzero, nonblank character in column six. Normally input is in the form accepted by *f77(1)*.

—t*n* Make the nonzero integer *n* the lowest valued label in the output program (default 10).

FILES

*/tmp/struct**	Temporary storage
*/usr/lib/struct/**	Various optional programs

CAVEATS

Struct knows FORTRAN 66 syntax, but not full FORTRAN 77.

If an input FORTRAN program contains identifiers, which are reserved words in Ratfor, the structured version of the program will not be a valid Ratfor program.

The labels generated cannot go above 32767.

If you get a **goto** without a target, try **—e**.

SEE ALSO

f77(1).

NAME

stty — set terminal options

SYNOPSIS

stty [*option...*]

DESCRIPTION

Stty sets certain I/O options on the current output terminal, placing its output on the diagnostic output (*stderr*). With no argument, it reports the speed of the terminal and the settings of the options which are different from their defaults. With the argument **all**, all normally used option settings are reported. With the argument **everything**, everything **stty** knows about is printed.

OPTIONS

The option strings are selected from the following set:

even
Allow even parity input.

—even
Disallow even parity input.

odd
Allow odd parity input.

—odd
Disallow odd parity input.

raw
Raw mode input (*no* input processing (**erase**, **kill**, **interrupt**, ...); parity bit passed back).

—raw
Negate raw mode.

cooked
Same as **—raw**.

cbreak
Make each character available to *read(2)* as received; no **erase** and **kill** processing, but all other processing (**interrupt**, **suspend**, ...) is performed.

—cbreak
Make characters available to **read** only when newline is received.

—nl
Allow carriage return for newline, and output CR–LF for carriage return or newline.

nl Accept only newline to end lines.

echo
Echo back every character typed.

—echo
Do not echo characters.

lcase
Map uppercase to lowercase.

—lcase
Do not map case.

tandem
Enable flow control, so that the system sends out the stop character when its internal queue is in danger of overflowing on input, and sends the start character when it is ready to accept further input.

—tandem
Disable tandem flow control.

dtr Enable DTR hardware flow control, so that the system asserts DTR when the internal queue is in danger of overflowing on input, and deasserts DTR when the queue is empty.

—dtr
Disable dtr flow control.

cts Enable CTS hardware flow control, so that data is transmitted only if the CTS pin is asserted.

—cts
Disable cts flow control.

—tabs
Replace tabs by spaces when printing.

tabs
Preserve tabs.

ek Set erase and kill characters to # and @.

gspeed
Sets up split baud rates such that input is received at 300 baud while output is sent at 9600 baud.

speed
Prints the current baud rate.

hup
When *stdout* associated with this process is closed for the last time, hang up the terminal.

For the following commands, which take a character argument *c*, you may also specify *c* as the **u** or **undef**, to set the value to be undefined. A value of ⟨CTRL–*x*⟩, a two–character sequence, is also interpreted as a control character, with ⟨CTRL–?⟩ representing delete.

erase *c*
Set erase character to *c* (default #, but often reset to ⟨CTRL–H⟩).

kill *c*
Set kill character to *c* (default @, but often reset to ⟨CTRL–U⟩).

intr *c*
Set interrupt character to *c* (default ⟨DEL⟩ or ⟨CTRL–?⟩ (delete), but often reset to ⟨CTRL–C⟩).

quit *c*
Set quit character to *c* (default ⟨CTRL–\⟩).

start *c*
Set start character to *c* (default ⟨CTRL–Q⟩).

stop *c*
Set stop character to *c* (default ⟨CTRL–S⟩).

eof *c*
Set end–of–file character to *c* (default ⟨CTRL–D⟩).

brk *c*
Set break character to *c* (default undefined). This character is an extra wakeup causing character.

cr0 cr1 cr2 cr3
Select style of delay for carriage return (see *ioctl(2)*).

nl0 nl1 nl2 nl3
Select style of delay for linefeed.

tab0 tab1 tab2 tab3
Select style of delay for tab.

ff0 ff1
Select style of delay for form feed.

bs0 bs1
Select style of delay for backspace.

dec
Set all modes suitable for Digital Equipment Corp. operating systems users; (**erase**, **kill**, and **interrupt** characters to ⟨CTRL–?⟩, ⟨CTRL–U⟩, and ⟨CTRL–C⟩, **decctlq** and **newcrt**).

tek Set all modes suitable for Tektronix 4014 terminal.

0 Hang up phone line immediately.

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600
Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH–11 interface.)

exta 19200 extb 38400
Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH–11 interface.)

A teletype driver with more functionality than the basic driver is fully described in *tty(4)*. The following options apply only to it:

new
Use new driver (switching flushes typeahead).

crt Set options for a CRT (**crtbs**, **ctlecho** and, if ⟩= 1200 baud, **crterase** and **crtkill**).

crtbs
Echo backspaces on erase characters.

prterase
For printing terminal **echo**–erased characters backwards within \ and /.

crterase
Wipe out erased characters with ⟨backspace–space–backspace⟩. (Note: If **crterase** is set, but **crtbs** is not, nothing will change. The tty driver must echo backspaces before it can perform erasure using backspaces.)

—crterase
Leave erased characters visible; just backspace.

crtkill
Wipe out input on like **kill** through **crterase.**

—crtkill
Just echo–line **kill** character and a newline on line–kill.

ctlecho
Echo control characters as ⟨CTRL–*x*⟩ (and delete as ⟨CTRL–?⟩). Print two backspaces following the EOT character (⟨CTRL–D⟩).

—ctlecho
Control characters echo as themselves; in cooked mode EOT (⟨CTRL–D⟩) is not echoed.

decctlq
After output is suspended (normally by ⟨CTRL–S⟩), only a start character (normally ⟨CTRL–Q⟩) will restart it. This is compatible with DEC's vendor supplied systems.

—decctlq
After output is suspended, any character typed will restart it; the start character will restart output without providing any input. (This is the default.)

tostop
Background jobs stop if they attempt terminal output.

—tostop
Output from background jobs to the terminal is allowed.

tilde
Convert ~ (tilde) to ' on output (for Hazeltine terminals).

—tilde
Leave poor ~ (tilde) alone.

flusho
Output is being discarded usually because user hit ⟨CTRL–O⟩ (internal state bit).

—flusho
Output is not being discarded.

pendin
Input is pending after a switch from **cbreak** to **cooked** and will be re–input when a **read** becomes pending or more input arrives (internal state bit).

—pendin
Input is not pending.

mdmbuf
Start/stop output on carrier transitions (not implemented).

—mdmbuf
Return error if **write** attempted after carrier drops.

litout
Send output characters without any processing.

—litout
Do normal output processing, inserting delays, and so forth.

nohang
Don't send hangup signal if carrier drops.

—nohang
Send hangup signal to control process group when carrier drops.

etxack
Diablo style **etx/ack** handshaking (not implemented).

The following special characters are applicable only to the new teletype driver and are not normally changed:

susp *c*
Set suspend process character to *c* (default ⟨CTRL–Z⟩).
dsusp *c*
Set delayed suspend process character to *c* (default ⟨CTRL–Y⟩).
rprnt *c*
Set reprint line character to *c* (default ⟨CTRL–R⟩).
flush *c*
Set flush output character to *c* (default ⟨CTRL–O⟩).
werase *c*
Set word erase character to *c* (default ⟨CTRL–W⟩).
lnext *c*
Set literal next character to *c* (default ⟨CTRL–V⟩).

EXAMPLES

The following invocation will set the users **tty** baud rate to 9600:

```
stty 9600
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

ioctl(2), tset(1), tty(4).

NAME

su — substitute userid temporarily

SYNOPSIS

su [—] [—**e**] [—**f**] [*username*]

DESCRIPTION

Su demands the password of the specified *username,* and if it is given, changes to that *username* and invokes the Shell *sh(1sh)* without changing the current directory. The user environment is unchanged except for **HOME** and **SHELL**, which are taken from the password file for the user being substituted (see *environ(7)*), unless the —**e** option is given. The new userid stays in force until the Shell exits.

If no *username* is specified, root is assumed. To remind the superuser of his or her responsibilities, the Shell substitutes # for its usual prompt.

When a user attempts to **su** to root (or any username with a userid of 0), the attempt is reported to the system via *syslog(3c)*. This applies to all attempts, including those where an invalid password is given.

OPTIONS

— Execute the shell as —**su**, causing the startup file (*.profile* or *.login*) to be read.

—**e** Use the original user's environment. With this option, the substituted user's powers are given, but the user uses current environment variables and aliases (only in *csh(1csh)).*

—**f** Execute the shell with the —**f** flag. (This only works with *csh(1csh)).*

VARIABLES

HOME	The home directory for the given username.
SHELL	The login shell for the given username.
USER	The given username.
TERM	The type of terminal being used.

RETURN VALUE

The exit code for a successful execution of **su** is that of the shell that was executed.

[1]	The username given is not in the password file.
[2]	The password given was incorrect.
[3]	The groupid for the given username could not be set.
[4]	The groups for the username could not be set.
[5]	The username could not be set.
[6]	Could not change directory to the home directory for the given username.
[7]	The shell program could not be executed.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

When substituting a username which has no execute or read permission in the current directory, the shell may not be able to execute commands from the startup file. **Su** can not check for this possibility, since some username's do not execute shells.

If the basename of the shell to be executed is not **csh**, the **—f** option is ignored.

SEE ALSO

csh(1csh), sh(1sh), environ(7).

NAME

subset — generate a subset of a vector

SYNOPSIS

subset [—**a***n*] [—**b***n*] [—**c***n*] [—**F***vector*] [—**i***n*] [—**l***n*] [—**nl**] [—**np**] [—**p***n*] [—**s***n*] [—**t***n*] [*vector ...*]

DESCRIPTION

Output is a vector of elements selected from the input based on a *key* and the values of several parameters set by command-line options. If no *vector* is given, the standard input is assumed.

If a *master* vector is given by the **-F** option, then the *key* for each element of the input is the corresponding element of *master*. Otherwise the *key* for each input element is the input element itself.

The input element is selected if the *key* is:

above the value specified by the **-a** option, *or*
below the value specified by the **-b** option, *or*
equal to the value specified by the **-p** (*pick*) option, *and*
not equal to the value specified by the **-l** (*leave*) option.

If neither **-a**, **-b**, nor **-p** is given, then the element is selected unless it is equal to the value specified by the **-l** option.

You can also select elements according to their *index* (position in the vector).

The **-s** (*start*), **-t** (*terminate*), and **-i** (*interval*) options select every **i**th element from element number **s** to element number **t**.

The **-nl** and **-np** options leave or pick elements whose index numbers are specified by the *master* vector. For example, if you use the **-np** option and the elements of the *master* are **1 3 9**, the first, third, and ninth elements of the input are selected.

If you use **-s**, **-t**, **-i**, **-nl**, or **-np** in conjunction with **-a**, **-b**, **-p**, or **-l**, only those elements meeting both the position and value qualifications are selected.

If you use either **-nl** or **-np** in conjunction with **-a**, **-b**, **-p**, or **-l**, the *key* for each element is the value of the element, not the value from the *master*.

OPTIONS

—**a***n*
select elements whose value is above *n*.

—**b***n*
select elements whose value is below *n*.

—**c***n*
n elements per output line.

—**F***vector*
vector is the *master*.

—i*n*
select every *n*th element. Default is 1 (every element).

—l*n*
leave (do not select) elements whose value is equal to *n*.

—nl
leave elements whose index is given in *master*.

—np
pick (select) elements whose index is given in *master*.

—p*n*
select elements whose value is equal to *n*.

—s*n*
select elements starting with the *n*th. Default is 1.

—t*n*
select elements up to the *n*th. Default is 32767.

EXAMPLES

The following example outputs the even elements of A.

```
subset -i2,s2 A
```

The following example outputs the elements of A whose values are greater than 8 or less than 3.

```
subset -a8,b3 A
```

For each element in B whose value is 1, the following example outputs the corresponding element of A.

```
subset -FB,p1 A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

sum — generate checksum and count 1024–byte blocks in a file

SYNOPSIS

sum [*filename* . . .]

DESCRIPTION

Sum calculates and prints a 16–bit checksum for the named file, and also prints the number of 1024–byte blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

EXAMPLES

The following example will print the checksum followed by a block count and the filename for each file in */da/tmp*:

```
sum /da/tmp/*
```

DIAGNOSTICS

Read error This response is indistinguishable from end–of–file on most devices; check the block count.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

wc(1).

NAME

stop, suspend — stop execution of jobs (**csh** built–in)

SYNOPSIS

stop %*job* . . .
suspend

DESCRIPTION

The command **stop** sends the stop signal to the named jobs. The resulting job(s) show up on the *jobs(1csh)* listing with the message "Stopped (signal)". The job can be restarted using the commands described in *fg(1csh)*.

The *suspend* command is used to suspend the current shell. This is required since the shell always ignores the stop signal. This is useful with shells started with *su(1)* or other subshells.

EXAMPLES

Assume that the command "make myprog" is job number 1, and is the only *make(1)* job running. The following shows two different ways to stop the job.

```
stop %1
stop %make
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the error message has occurred.

CAVEATS

The **suspend** command is not present in *sh(1sh)*. This is no problem, since the Bourne shell does not ignore the stop signal.

There is no way to suspend a login shell.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), make(1), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), kill(2), killpg(2), signal(3c).

NAME

symorder — rearrange name list

SYNOPSIS

symorder *orderlist symbolfilename*

DESCRIPTION

Orderlist is a file containing symbols to be found in *symbolfilename*, one symbol per line.

Symbolfilename is updated in place to put the requested symbols first in the symbol table, in the order specified. This is done by swapping the old symbols in the required spots with the new ones. If all of the order symbols are not found, an error is generated.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

nlist(3c), a.out(5).

NAME

tail — copy the last part of a file to standard output

SYNOPSIS

tail [± [*number*] [**lbc**] [**fr**]] [*filename*]

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.

Specifying **r** causes **tail** to print lines from the end–of–file in reverse order. The default for **r** is to print the entire file this way. Specifying **f** causes **tail** to not quit at end–of–file, but rather wait and try to read repeatedly in hopes that the file will grow. If both **r** and **f** are given, the **f** is ignored.

OPTIONS

—f Causes **tail** to not quit at end–of–file, but rather wait and try to read repeatedly in hopes that the file will grow.

—r Causes tail to print lines from the end of the file in reverse order.

+*number*
Copying begins at distance + *number* from the beginning of the input.

—*number*
Copying begins at distance —*number* from the end of the input. *Number* is counted in units of lines, 1024–byte blocks or characters, according to the appended option **l, b** or **c,** respectively. When no units are specified, counting is by lines. When the size argument is not specified, the last 10 lines are copied.

EXAMPLES

The following example reads the output from *nroff(1)* which is being written to the file *text.out*. The last 30 lines of output are displayed first, and then all subsequent output is displayed.

```
nroff file > text.out&
tail -30lf text.out
```

This example builds a new copy of the file *example* containing all but the first 29 lines of the original copy.

```
tail +30 example > tmp
mv tmp example
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

When the *—number* option is used, a maximum of 16384 characters can be copied.

Various kinds of anomalous behavior may happen with character special files.

SEE ALSO

dd(1), head(1).

NAME

talk — talk to another user

SYNOPSIS

talk *person*[*ttyname*]

DESCRIPTION

Talk is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on you own machine, then *person* is just the person's loginname. If you wish to talk to a user on another host, then *person* is of the form :

host!user or
host.user or
host:user or
user@host

Note that *user@host* is preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, it sends the message

```
Message from TalkDaemon@his_machine...  talk: connection
requested by your_name@your_machine.  talk: respond with:
talk your_name@your_machine
```

to the user you wish to talk to. At this point, the recipient of the message should reply by typing

```
talk  your_name@your_machine
```

It doesn't matter from which machine the recipient replies, as long as his or her loginname is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing <CTRL–L> will cause the screen to be reprinted, while your **erase**, **kill**, and **word kill** characters will work in **talk** as normal. To exit, just type your interrupt character; **talk** then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the **mesg** command. At the outset talking is allowed. Certain commands, in particular **nroff** and *pr(1)* disallow messages in order to prevent messy output.

FILES

/etc/hosts	To find the recipient's machine.
/etc/utmp	To find the recipient's **tty**.

RETURN VALUE

[0]	No errors.
[nonzero]	Errors occurred.

SEE ALSO

mesg(1), who(1N), mail(1), write(1).

NAME

tar — tape archiver

SYNOPSIS

tar *key* [*tapefilename*] [*blocksize*] [*name ...*]

DESCRIPTION

Tar saves and restores multiple files on a single file (usually a magnetic tape, but it can be any file). **Tar**'s actions are controlled by the *key* argument. The *key* consists of one (or possibly two) "function" letter(s) and possibly one or more function modifiers. These are listed in the OPTIONS section. The *key* may be preceded by a dash (–), but this is optional.

The *tapefilename* argument is given with the **f** option. If this is a dash (–), standard input or standard output is used for input or output. The *blocksize* argument is given only if the **b** option is given.

The *name* arguments are the names of the files and directories to be archived. The names may be preceded by a **—C,** which specifies that **tar** is to *chdir(2)* (change directories) to that directory name (this is useful when the directories to be archived do not have a close common parent and it is undesirable to save a large directory structure). All arguments following that name until the next **—C** is taken to be relative to that name. The name following the **—C** is not archived. If the name begins with a slash (/), **tar** also does a **chdir** to the directory, but only to archive that directory. After that, the current directory is changed back to what it was before.

Directories are recursively archived, meaning that the contents of the directory, its subdirectories, and so forth are all archived.

Previous restrictions dealing with **tar**'s inability to properly handle blocked archives have been lifted.

OPTIONS

The function portion of the key is specified by one of the following letters (Exception: both **c** and **r** may be given, since **c** implies **r)** :

c Create a new tape; writing begins on the beginning of the tape instead of after the last file. This command implies **r**.

r The named files are written on the end of the tape. The **c** function implies this. (NOTE: Some streaming tape drives cannot support this function. In this case, an error message will be printed to this effect.)

t The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.

u The named files are added to the tape if either they are not already there or have been modified since last put on the tape. (NOTE: Some streaming tape drives can not support this function. In this case, an error message will be printed to this effect.)

x The named files are extracted from the tape. If the 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 file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier.

The following are function modifiers:

b **Tar** uses the next argument as the number of 512 byte blocks to use for tape records. The default is 20. This option should only be used with raw magnetic tape archives (See **f** below). The block size is determined automatically when reading tapes (function letters **x** and **t**).

f **Tar** uses the next argument as the name of the archive instead of */dev/rmt?*. If the name of the file is a dash (—), **tar** writes to standard output or reads from standard input, whichever is appropriate. Thus, **tar** can be used as the head or tail of a filter chain. **Tar** can also be used to move hierarchies with the command

```
cd fromdir; tar cf - . ¦ (cd todir; tar xf -)
```

h Force **tar** to follow symbolic links as if they were normal files or directories. Normally, **tar** does not follow symbolic links.

i **Tar** stores a *checksum* for each directory in order to check for **read/write** errors and corrupted tapes. Normally, a *checksum* error will cause **tar** to terminate. The **i** option tells **tar** to ignore directory *checksum* errors. These files will be ignored.

l Tells **tar** to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.

m Tells **tar** not to restore the modification times. The modification time will be the time of extraction.

o On output, **tar** normally places information specifying owner and modes of directories in the archive. Former versions of **tar**, when encountering this information will give error message of the form

```
⟨name⟩/: cannot create.
```

This option will suppress the directory information.

p This option says to restore files to their original modes, ignoring the present *umask(2)*. **Setuid** and sticky information will also be restored to the superuser.

v Normally **tar** does its work silently. The **v** (verbose) option makes **tar** print the name of each file it treats preceded by the function letter (in the case of **c** and **r**, the letter *a* says that the file was added, and *r* says that the file was replaced). With the **t** function, the verbose option gives more information about the tape entries than just their names. All information is printed on the standard error, so

redirection of output must take this into account.

w **Tar** prints the action to be taken followed by filename, then waits for user confirmation. If a word beginning with *y* is given, the action is done. Any other input means don't do it.

B Forces input and output blocking to 20 blocks per record. This option was added so that **tar** can work across a communications channel where the block may not be maintained.

F If given once, any directories named *SCCS* and any files named *core* or *errs* are ignored. If given twice, any files named *a.out* or having the suffix *.o* are also ignored.

T Take list of filenames and **—C** commands from standard input.

W Like **w**, but produces a very terse description of the action to be taken.

0, ..., 9
This modifier selects an alternate drive on which the tape is mounted. The default is */dev/rmt8*.

EXAMPLES

To archive files from */usr/include* and from */etc*, you might use one of the following:

```
tar c -C /usr include -C / etc
tar c /usr/include /etc
```

To archive all files in a directory, you could use the following:

```
find . -print ¦ tar Tc
```

FILES

/dev/rmt8	The default input/output archive file.
*/tmp/tar**	Temporary file for the **u** function.

CAVEATS

The **r** and **u** functions may not be available with some tape drives, such as streaming tape drives.

In order to keep streaming tapes streaming, it is recommended that a blocking factor (for the **b** modifier) of 256 be given. This will cause **tar** to buffer the data in 128 kbyte blocks.

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 current limit on file name length is 100 characters. If longer file names are required and portability to other UNIX systems is not required, see *cpio(1)*.

There is no way to selectively follow symbolic links.

SEE ALSO

ar(1), cpio(1), cpio(5), tar(5), dump(8), restore(8).

NAME

tbl — format tables for nroff or troff

SYNOPSIS

tbl [**—TX**] [*filenames*]

DESCRIPTION

Tbl is a preprocessor that formats tables for **nroff(1)** or **troff(1).** The input files are copied to the standard output, except for lines between **.TS** and **.TE** command lines, which are assumed to describe tables and are reformatted by **tbl.** (The **.TS** and **.TE** command lines are not altered by **tbl.**)

.TS is followed by global options. The available global options are:

center	Center the table (default is left–adjust).
expand	Make the table as wide as the current line length.
box	Enclose the table in a box.
doublebox	Enclose the table in a double box.
allbox	Enclose each item of the table in a box.
tab (*x*)	Use the character *x* instead of a tab to separate items in a line of input data.

The global options, if any, are terminated with a semi–colon (;).

The 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 widths, 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; unit 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 pipe 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 (for example, line printers).

If no filenames are given as arguments (or if — is specified as the last argument), **tbl** reads the standard input, so it may be used as a filter. When it is used with **eqn**(1) or **neqn**, **tbl** should come first to minimize the volume of data passed through pipes.

EXAMPLES

If we let ← represent a tab (which should be typed as a genuine tab), then the input:

```
.TS
center box ;
cB s s
cI | cI s
^ | c c
I | n n .
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
.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

CAVEATS

See *CAVEATS* under *nroff(1)*.

SEE ALSO

eqn(1), mm(1), mmt(1), nroff(1), troff(1), mm(7), mv(7).

REFERENCES

Table Formatting Program (Tbl) in the *UTek Tools* documentation.

NAME

td — display GPS on a Tektronix terminal

SYNOPSIS

td [**—e**] [**—g**] [**—r***n*] [**—u**] [*GPS file ...*]

DESCRIPTION

Output is scope code for a Tektronix 4014 terminal or 4100 series bitmapped color terminal. A viewing window is computed from the maximum and minimum points in the first *file* unless *options* are provided. If no *file* is given, the standard input is assumed.

To use color and hardware text on the 4100 series terminal, the **$TERM** environment variable must be set appropriately.

OPTIONS

—e Do not erase screen before initiating display.

—g Do not use hardware text (4100 series only).

—r*n*
Window on GPS region *n*, *n* between 1 and 25 inclusive.

—u Window on the entire GPS universe.

EXAMPLES

The following example displays A.g and B.g. The viewing window is built to include all of A.g.

```
td A.g B.g
```

VARIABLES

TERM The user's terminal type.

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

tee — redirect to standard output and files (pipe fitting)

SYNOPSIS

tee [**—i**] [**—a**][*filename...*]

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *filenames*. Option **—i** ignores interrupts; option **—a** causes the output to be appended to the *filenames* rather than overwriting them.

OPTIONS

—a Causes the output to be appended to the *filenames* rather than overwriting them.

—i Ignores interrupts.

EXAMPLES

The following example writes the output from **who** on the file *whois* as well as printing it on the terminal:

```
who ¦ tee whois ¦ more
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

csh(1csh), sh(1sh).

NAME

tekset — send reset characters for Tektronix 4014 terminal

SYNOPSIS

tekset

DESCRIPTION

Tekset resets the Tektronix 4014 display terminal. It clears the display screen and sets the display mode to alpha and the characters to the smallest font.

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

telnet — user interface to the TELNET protocol

SYNOPSIS

telnet [*host* [*port*]]

DESCRIPTION

Telnet is used to communicate with another host using the TELNET protocol. If **telnet** is invoked without arguments, it enters command mode, indicated by its prompt, **telnet**>. In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an *open* command (see below) with those arguments.

Once a connection has been opened, **telnet** enters input mode. In this mode, text typed is sent to the remote host. To issue **telnet** commands when in input mode, precede them with the **telnet** escape character (initially CTRL–], control right–bracket). When in command mode, the normal terminal editing conventions are available.

The following commands are available. Only enough of each command to uniquely identify it needs to be typed.

open *host* [*port*]
Open a connection to the named host. If the no–port number is specified, **telnet** will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see *hosts(5n)*) or an Internet address specified in the dot notation.

close Close a TELNET session and return to command mode.

quit Close any open TELNET session and exit **telnet**.

z Suspend **telnet**. This command only works when the user is using the *csh(1csh)*.

escape [*escape-char*]
Set the **telnet** escape character. Control characters may be specified as ^*x* where the single letter *x* is the control letter; for example, control–P is ^P.

status Show the current status of **telnet**. This includes the peer one is connected to, as well as the state of debugging.

options Toggle viewing of TELNET options processing. When options viewing is enabled, all TELNET option negotiations will be displayed. Options sent by **telnet** are displayed as SENT, while options received from the TELNET server are displayed as RCVD.

crmod Toggle carriage return mode. When this mode is enabled any carriage return characters received from the remote host will be mapped into a carriage return and a linefeed. This mode does not affect those characters typed by the user, only those received. This mode is not very useful, but is required for some

hosts that like to ask the user to do local echoing.

? [*command*]
Get help. With no arguments, **telnet** prints a help summary. If a command is specified, **telnet** will print the help information available about the command only.

CAVEATS

This implementation is very simple because *rlogin(1n)* is the standard mechanism used to communicate locally with hosts.

SEE ALSO

csh(1csh), rlogin(1n), hosts(5n).

NAME

test — condition command

SYNOPSIS

test *expr*
[*expr* **]**

DESCRIPTION

Test evaluates the expression *expr*, and if its value is true then returns zero exit status; otherwise, a nonzero exit status is returned. **Test** returns a nonzero exit if there are no arguments.

If **test** is executed as the bracket [, the last argument must be the matching bracket].

The command is available as a builtin **sh** command and as a shell script for use in **csh** and **make**.

The following primitives are used to construct *expr*:

—r *filename*
True if the file exists and is readable.

—w *filename*
True if the file exists and is writable.

—x *filename*
True if the file exists and is executable.

—f *filename*
True if the file exists and is not a directory.

—d *filename*
True if the file exists and is a directory.

—c *filename*
True if the file exists and is a character special file.

—b *filename*
True if the file exists and is a block special file.

—U *filename*
True if the file exists and is a UTek–domain socket.

—u *filename*
True if the file exists and its set–user–ID bit is set.

—g *filename*
True if the file exists and its set–group–ID bit is set.

—k *filename*
True if the file exists and its sticky bit is set (see *chmod(2)*).

—S *filename*
True if the file exists and is a symbolic link.

—s *filename*
True if the file exists and has a size greater than 0.

—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 0.

—n *s1*
True if the length of the string *s1* is nonzero.

s1 = *s2*
True if the strings *s1* and *s2* are equal.

s1 **!=** *s2*
True if the strings *s1* and *s2* are not equal.

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**, or **—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

(*expr* **)**
Parentheses for grouping

There are four functions which return integer values and may only be used in algebraic comparisons.

—l *string*
Length of *string*

—C *filename*
Time of the last status change to *filename* (see *stat(2))*

—M *filename*
Time of the last modification to *filename* (see *stat(2))*

—A *filename*
Time of the last access to *filename* (see *stat(2))*

EXAMPLES

The following example shows a piece of a shell script which prints a usage message if the first argument to the script is null:

```
if test -z "$1"
then
        echo "usage : script arg"
fi
```

The following shell script prints a message for each of its arguments that are longer than 72 characters:

```
#!/bin/sh
for i in "$@"
do
        if [ -l $i -gt 72 ]
        then
                echo "Argument too long :" $i
        fi
done
```

This example shows a use for the **—M** option. If the file *copy* is older than the file *original*, the latter is copied to *copy*:

```
if test -M copy -lt -M original
then
        cp original copy
fi
```

RETURN VALUE

[0]	The expression evaluates to true.
[1]	The expression evaluates to false.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

The **—a** option has higher precedence than **—o.** Notice that all the operators and flags are separate arguments to **test**.

Notice also that parentheses are meaningful to the Shell and must be escaped.

The file access operators **—r**, **—w**, and **—x** always evaluate to false if the filename is null, as in the following example:

```
test -r ""
```

Test is actually a shell script which executes the **sh** builtin command. This is for use in programs such as *make(1)* that may need to execute **test** directly.

SEE ALSO

csh(1csh), expr(1), find(1), make(1), sh(1sh), test(1sh), access(2), chmod(2), stat(2).

NAME

test — condition command (*sh* built–in)

SYNOPSIS

test *expr*
[*expr* **]**

DESCRIPTION

Test evaluates the expression *expr*, and if its value is true then returns 0 exit status; otherwise, a nonzero exit status is returned. **Test** returns a nonzero exit if there are no arguments.

If **test** is executed as a bracket [, the last argument must be the matching bracket].

The command is available as a builtin **sh** command and as a shell script for use in **csh** and **make**.

The following primitives are used to construct *expr*:

—r *filename*
True if the file exists and is readable.

—w *filename*
True if the file exists and is writable.

—x *filename*
True if the file exists and is executable.

—f *filename*
True if the file exists and is not a directory.

—d *filename*
True if the file exists and is a directory.

—c *filename*
True if the file exists and is a character special file.

—b *filename*
True if the file exists and is a block special file.

—U *filename*
True if the file exists and is a UTek–domain socket.

—u *filename*
True if the file exists and its set–user–ID bit is set.

—g *filename*
True if the file exists and its set–group–ID bit is set.

—k *filename*
True if the file exists and its sticky bit is set (see *chmod(2)*).

—S *filename*
True if the file exists and is a symbolic link.

—s *filename*
True if the file exists and has a size greater than 0.

—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 0.

—n *s1*
True if the length of the string *s1* is nonzero.

s1 = *s2*
True if the strings *s1* and *s2* are equal.

s1 **!=** *s2*
True if the strings *s1* and *s2* are not equal.

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**, or **—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

(*expr* **)**
Parentheses for grouping

There are four functions which return integer values and may only be used in algebraic comparisons:

—l *string*
Length of *string*

—C *filename*
Time of the last status change to *filename* (see *stat(2))*

—M *filename*
Time of the last modification to *filename* (see *stat(2))*

—A *filename*
Time of the last access to *filename* (see *stat(2))*

EXAMPLES

The following example shows a piece of a shell script which prints a usage message if the first argument to the script is null:

```
if test -z "$1"
then
        echo "usage : script arg"
fi
```

The following shell script prints a message for each of its arguments that are longer than 72 characters:

```
#!/bin/sh
for i in "$@"
do
        if [ -l $i -gt 72 ]
        then
                echo "Argument too long :" $i
        fi
done
```

This example shows a use for the **—M** option. If the file *copy* is older than the file *original*, the latter is copied to *copy*:

```
if test -M copy -lt -M original
then
        cp original copy
fi
```

RETURN VALUE

[0]	The expression evaluates to true.
[1]	The expression evaluates to false.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

The **—a** option has higher precedence than **—o.** Notice that all the operators and flags are separate arguments to **test**.

Notice also that parentheses are meaningful to the Shell and must be escaped.

The file access operators **—r**, **—w**, and **—x** always evaluate to false if the filename is null, as in the following example:

```
test -r ""
```

The builtin shell version of **test** does not print error message tags along with error messages.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), expr(1), find(1), hash(1sh), login(1), make(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), access(2), chmod(2), execve(2), stat(2).

NAME

tftp — DARPA Trivial File Transfer Protocol client program

SYNOPSIS

tftp [**t–name**] [*port*]

DESCRIPTION

Tftp is a program which connects to a server *(tftpd(8n))* which supports the DARPA Trivial File Transfer Protocol on a remote host specified by **host–name.**

The use of **tftp** does not require an account or password on the remote system. Due to the lack of authentication information, **tftpd(8n)** allows only publicly readable files to be accessed. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling the **tftp** service.

OPTIONS

host-name

This is the name of the remote host you want to connect to.

port

Use this port instead of the one in *services(5n)*.

EXAMPLES

Invoke **tftp**:

```
tftp NotHereBox
```

If the host NotHereBox is known, you receive a prompt for further **tftp** commands.

```
tftp›
```

Type any of the following commands:

connect

Followed by a hostname, allows you to initiate a connection to another host.

mode

Set according the type of data you wish to send. Available settings are *ascii, binary,* and *mail*

put

Followed by a file name, sends the file to the remote system.

get

Followed by a file name, gets the named file from the remote system.

quit

exit tftp

verbose
Toggles verbose mode. Gives information about files transferred.

trace
Toggles on or off generation of a message every time a packet is transferred.

status
Shows current status of connection, which mode data is sent in, what host you are connected to, etc.

rexmt
Followed by a number of seconds value. Set the per-packet retransmission timeout. Tftp expects an acknowledgement after each packet is sent. This command lets you set how many seconds before **tftp** gives up and retries a packet.

timeout
Followed by a number of seconds integer value. Set the total retransmission timeout. When this time is up **tftp** prints out " Transmission Timed Out " and then prompts you for another command.

? Print out help information.

Suppose you want to send file "myfile" to NotHereBox. If "myfile" is an ASCII file you do not need to use the mode command to change the data transmission mode because the default is ASCII.

```
tftp› put myfile
```

Use the quit command to exit tftp.

```
tftp› quit
```

RETURN VALUE

[0] No errors encountered.

[1] Errors encountered.

[3] System socket error

CAVEATS

This program is known only to be self consistent (that is, it operates with the server TFTP program, **tftpd(8n)**).

The search permissions of the directories leading to the files accessed are not checked.

SEE ALSO

services(5n) tftpd(8n).

NAME

time — time a command

SYNOPSIS

time *command*

DESCRIPTION

The given command is executed; after it is complete, **time** prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on the diagnostic output stream (standard error).

The signals interrupt, quit, and terminate are ignored.

EXAMPLES

The following example runs the command *ls(1)* with some options and prints the amount of time it took to run the command:

```
time ls -al
```

RETURN VALUE

The exit status returned is that of the command being timed. If the command does not exist, the exit status is NP_ERR.

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Elapsed time is accurate to the second, while the CPU times are measured to the 60th second. Thus, the sum of the CPU times can be up to a second larger than the elapsed time.

SEE ALSO

ls(1), ctime(3c).

NAME

time — print time statistics (**csh** built–in)

SYNOPSIS

time [*command* [*args...*]]

DESCRIPTION

With no arguments, **time** prints a summary of the times used by the current shell and all of its child processes. With a *command* argument, the command is executed and the time summary is printed for that command execution.

The *time* shell variable may be set to contain a minumum amount of CPU time used by a process to cause a time summary to be printed automatically, and a summary format.

The summary format is controlled by the value of the second element of the *time* variable. The format is a string which may contain text, and the following special value identifiers, which cause the given values to be printed.

%U	user time
%S	system time
%E	elapsed time
%P	average percentage of CPU cycles used
%W	number of swaps
%X	text segment resident set size (in kilobytes)
%D	data segment resident set size (in kilobytes)
%K	total resident set size (in kilobytes)
%M	maximum resident set size (in kilobytes)
%F	major page faults
%R	minor page faults
%I	blocks read
%O	blocks written

The default summary format is ''%Uu %Ss %E %P %X + %Dk %I + %Oio %Fpf + %Ww''.

The *command* must be a simple command, not an alias or control statement. If necessary, a separate shell is created to print the summary.

EXAMPLES

The following example runs the command "make myprog" and prints a time summary after the command has completed.

```
time make myprog
```

The following setting of the *time* variable will cause all processes that use more than 5 seconds of CPU time to have an automatic time summary printed. The summary will contain the user, system, and elapsed times.

```
set time=(5 "%U user  %S system  %E elapsed")
```

RETURN VALUE

The return value is the value returned by the command executed (0 if none given), or 1 if the command does not exist.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1), times(1sh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), getitimer(2), setitimer(2), getrusage(2).

NAME

times — print accumulated user and system time for shell processes (**sh** built–in)

SYNOPSIS

times

DESCRIPTION

Prints the accumulated user and system times for processes run from the shell.

EXAMPLES

The following is an example of the output produced by **times**:

```
1m3s 1m48s
```

This means that the total user time for processes run by the current shell is one minute and three seconds. The system time used is one minute and 48 seconds.

RETURN VALUE

[NO_ERRS] Command completed without error.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), time(1csh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2).

NAME

tip, cu — connect to a remote system

SYNOPSIS

tip [**—v**] [*—speed*] *system-name*
tip [**—v**] [*—speed*] *phone-number*
cu *phone-number* [**—t**] [**—s** *speed*] [**-a** *acu*] [**—l** *line*] [*—#*]

DESCRIPTION

Tip and **cu** establish a full–duplex connection to another machine, giving the appearance of being logged in directly on the remote CPU. It goes without saying that you must have a login on the machine (or equivalent) to which you wish to connect. The preferred interface is **tip**. The **cu** interface is included for those people attached to the **call unix** command of version 7 UNIX. This manual page describes only **tip**.

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well). A tilde (~) appearing as the first character of a line is an escape signal; the following are recognized:

~⟨CTRL–D⟩~.
: Drop the connection and exit (you may still be logged in on the remote machine).

~**c** [*name*]
: Change directory to *name* (no argument implies change to your home directory).

~**!**
: Escape to a shell; (exiting the shell will return you to **tip**).

~⟩
: Copy file from local to remote. **Tip** prompts for the name of a local file to transmit.

~⟨
: Copy file from remote to local. **Tip** prompts first for the name of the file to be sent, then for a command to be executed on the remote machine.

~**p** *from* [*to*]
: Send a file to a remote UTek or UNIX host. The **put** command causes the remote UTek or UNIX system to run the command string **cat** ⟩ *to*, while **tip** sends it the *from* file. If the *to* file isn't specified, the *from* filename is used. This command is actually a UTek (UNIX) specific version of the ~⟩ command.

~**t** *from* [*to*]
Take a file from a remote UTek or UNIX host. As in the **put** command, the *to* file defaults to the *from* filename if it isn't specified. The remote host executes the command string

```
cat from;echo ⟨CTRL-A⟩
```

to send the file to **tip**.

~**|** Pipe the output from a remote command to a local UTek process. The command string sent to the local UTek system is processed by the shell.

~**#** Send a **BREAK** to the remote system.

~**s** Set a variable (see the discussion below).

~⟨CTRL-Z⟩
Stop **tip** (only available with job control).

~**{** Receive a text file from the remote host using the XMODEM protocol. Must issue the appropriate command to start XMODEM transfer *before* giving this escape to *tip*. Translation is performed from CP/M file format (CR/LF) to UTek text file format (LF). If **beautify** is set then all bytes have the parity bit removed for consistency with UTek editors.

~**}** Send a text file to the remote host using the XMODEM protocol. Translation is done from UTek text file format to CP/M format as dictated by the protocol. Must issue the XMODEM command on the remote host first.

~**(** Receive a binary file from the remote host using the XMODEM protocol. No translation is performed.

~**)** Send a binary file to the remote host using the XMODEM protocol. No translations are done, the file is sent as is. The protocol dictates that the last 128 byte sector be padded with control Z characters, so this may not be suitable for transfer between UTek/UNIX hosts.

~**?** Get a summary of the tilde escapes.

Tip uses the file */etc/remote* to find how to reach a particular system and to find out how it should operate while talking to the system; refer to *remote(5n)* for a full description. Each system has a default baud rate with which to establish a connection. If this value is not suitable, the baud rate to be used may be specified on the command line; for example, **tip -300 mds**.

When **tip** establishes a connection it sends out a connection message to the remote system; the default value, if any, is defined in */etc/remote*.

When **tip** prompts for an argument (for example, during setup of a file transfer) the line typed may be edited with the standard **erase** and **kill** characters. A null line in response to a prompt, or an interrupt, will abort the dialogue and return you to the remote machine.

Tip guards against multiple users connecting to a remote system by opening modems and terminal lines with exclusive access, and by honoring the locking protocol used by *uucp(1n)*.

During file transfers **tip** provides a running count of the number of lines transferred. When using the ~> and ~< commands, the **eofread** and **eofwrite** variables are used to recognize end-of-file when reading, and specify end-of-file when writing (see below). File transfers normally depend on tandem mode for flow control. If the remote system does not support tandem mode, **echocheck** may be set to indicate **tip** should synchronize with the remote system on the **echo** of each transmitted character.

When **tip** must dial a phone number to connect to a system, it will print various messages indicating its actions. **Tip** supports the Racal-Vadic 831 auto-call-units; the Ventel 212+, Racal-Vadic 3451, and Bizcomp 1031 and 1032 integral call unit/modems.

INSTALLATION

Files needed are:

Name	Permissions	Owner
/bin/tip	-rwsr-xr-x	uucp
/etc/remote	-rw-rw-r--	sys
/usr/spool/uucp	drwxr-xr-x	uucp

Optional files are:

/etc/phones	-rw-rw-r--	sys
~/.tiprc		
~/.tipphones		
~/.tipremote		

FOR DIRECT RS-232-C CONNECTION

You need the following equipment:

An RS-232-C line to a target system. (Make sure you are able to login to the target system.)

A modem adapter cable (Tek part # 012-1120-00). It has two male ends that you plug into the female connectors on the target system and the workstation. It also switches the control lines so that two DCE ports can communicate as if they were a DCE/DTE pair.

Things to do:

1. Disable logins on the port you are going to use on your workstation by changing the entry in the */etc/ttys(5)* file. Then restart the init process by rebooting or sending it a hangup signal. For example make the following changes in the */etc/ttys* file on your workstation:

change: 1ytty1 to 0ytty1

2. Type: **kill -1 1**. This sends the init process the hangup signal. *Init(8)* rereads the */etc/ttys* file and turns off the login on port tty1. This prevents a **login** process from interfering with the port you are about to use for **tip**.

3. Use **chown** to give the tty port you are going to use for the **tip** connection to uucp. For example:

/etc/chown uucp /dev/tty1

4. Put an entry in the */etc/remote* file that describes the port you are going to use. For example:

```
direct¦direct 9600 baud line:\
    :dv=/dev/tty1:br#9600:ta:ie=^A\
    :oe=^A
```

5. Using the modem adapter cable, connect the login line from the target machine to the port you have chosen on your workstation.

6. Type **tip direct**, and **tip** will open a 9600 baud connection to the target host if you use the above examples.

FOR MODEM CONNECTION:

You need the following equipment:

A modem adapter cable as above.

A modem that is supported by the uucp Automatic Calling Unit library. Modems currently supported include: BIZCOMP, VENTEL 212+, VADIC 831 RS-232-C adaptor, VADIC 3451, HAYES smart modem. Other modems will work if they have an emulation mode for one of the above modems.

To make the connection perform the following steps:

1. Disable logins on the port you are going to use on your workstation by changing the entry in the */etc/ttys* file. Then restart the **init** process by rebooting or sending it a hangup signal. For example make the following changes in the */etc/ttys* file on your workstation. This example is for tty1:

change **1ytty1** to **0ytty1**

2. Use **chown** to give the tty port you are going to use for the **tip** connection to uucp. For example:

/etc/chown uucp /dev/tty1

3. Put an entry in the */etc/remote(5n)* file that describes the port and modem you are going to use. For example:

```
dial1200|1200 Baud Hayes :\
    :dv=/dev/tty1:br#1200:du:at=hayes:
```

4. Connect your modem to your workstation using the modem adapter cable.

5. Invoke **tip**. For example:

tip dial1200 5551212

Sometimes you need to type a carriage return after the "connected" message to get a prompt.

OPTIONS

—v This option causes **tip** to display the setting of its variables as they are done.

FILES

/etc/remote	Global system descriptions.
/etc/phones	Global phone number database.
${REMOTE}	Private system descriptions.
${PHONES}	Private phone numbers.
~/.tiprc	Initialization file.
*/usr/spool/uucp/LCK..**	Lock file to avoid conflicts with **uucp**.

DIAGNOSTICS

These are the most common messages. There are many others.

link down **Tip** displays this message when it cannot open the RS-232-C port. This will happen if a cable is not plugged in or if the cable that is used does not have the *carrier detect* pin connected.

all ports busy This message is displayed when a lock file is present in the */usr/spool/uucp* directory for the port **tip** is trying to use. This means that some other user is using this port and **tip** is locked out for the time being. If a lock file is present inadvertently, and there really is no one else trying to use this port, remove the lock file. You will need to run as superuser to do this.
Example:

```
rm /usr/spool/uucp/LCK..tty1
```

VARIABLES

Tip maintains a set of *variables* which control its operation. Some of these variable are read–only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the **s** escape. The syntax for variables is patterned after *vi(1)* and *mail(1mh)*. Supplying **all** as an argument to the set command displays all variables readable by the user. Alternatively, the user may request display of a particular variable by attaching a ? to the end. For example, **escape?** displays the current escape character.

Variables are numeric, string, character, or Boolean values. Boolean variables are set merely by specifying their name; they may be reset by prepending a ! to the name. Other variable types are set by concatenating an equal sign(=) and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables. Variables may be initialized at run–time by placing set commands (without the ~**s** prefix in a file *.tiprc* in your home directory). The **—v** option causes **tip** to display the sets as they are made. Certain common variables have abbreviations. The following is a list of common variables, their abbreviations, and their default values:

beautify
(Bool) Discard unprintable characters when a session is being scripted; abbreviated **be**.

baudrate
(num) The baud rate at which the connection was established; abbreviated **ba**.

dialtimeout
(num) When dialing a phone number, the time (in seconds) to wait for a connection to be established; abbreviated **dial**.

echocheck
(Bool) Synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted; default is **off**.

eofread
(str) The set of characters which signify and end–of–transmission during a ~< file transfer command; abbreviated **eofr**.

eofwrite
(str) The string sent to indicate end–of–transmission during a ~> file transfer command; abbreviated **eofw**.

eol
(str) The set of characters which indicate an end–of–line. **Tip** will recognize escape characters only after an end–of–line.

escape
(char) The command prefix (escape) character; abbreviated **es**; default value is a tilde (~).

exceptions
(str) The set of characters which should not be discarded due to the beautification switch; abbreviated **ex**; default value is \t\n\f\b.

force
(char) The character used to force literal data transmission; abbreviated **fo**; default value is ⟨CTRL–P⟩.

framesize
(num) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated **fr**.

host
(str) The name of the host to which you are connected; abbreviated **ho**.

prompt
(char) The character which indicates an end–of–line on the remote host; abbreviated **pr**; default value is \n. This value is used to synchronize during data transfers. The count of lines transferred during a file transfer command is based on receipt of this character.

raise
(Bool) Uppercase mapping mode; abbreviated **ra**; default value is **off**. When this mode is enabled, all lowercase letters will be mapped to uppercase by **tip** for transmission to the remote machine.

raisechar
(char) The input character used to toggle uppercase mapping mode; abbreviated **rc**; default value is ⟨CTRL-A⟩.

record
(str) The name of the file in which a session script is recorded; abbreviated **rec**; default value is **tip.record**.

script
(Bool) Session scripting mode; abbreviated **sc**; default is **off**. When **script** is **true**, **tip** will record everything transmitted by the remote machine in the script record file specified in **record**. If the **beautify** switch is on, only printable ASCII characters will be included in the script file (those characters between 040 and 0177) [also on XMODEM text file receives].

tabexpand
(Bool) Expand tabs to spaces during file transfers; abbreviated **tab**; default value is **false**. Each tab is expanded to eight spaces.

verbose
(Bool) Verbose mode; abbreviated **verb**; default is **true**. When verbose mode is enabled, **tip** prints messages while dialing, shows the current number of lines transferred during a file transfer operations, and more.

SHELL
(str) The name of the shell to use for the ~**!** command; default value is */bin/sh*, or taken from the environment.

HOME
(str) The home directory to use for the ~**c** command; default value is taken from the environment.

EXAMPLES

This is how variables are set up in the */etc/remote* file:

```
direct¦direct 9600 baud line:\
    :dv=/dev/tty1:br#9600:ta:ie=^A\
    :oe=^A
```

The *ie* and *oe* strings refer to the **eofread** and **eofwrite** strings described above. See the *remote(5n)* man page for more details.

This is how variables are set up using a tilde *s* escape. When **tip** answers "W~[set]" it is printed over your ~**s**. The escape character must be the first character typed on a line.

You type:

$~s

tip types:

~[set]

You type:

eofr=end_of_file_read_string

RETURN VALUE

[0]	No errors.
[nonzero]	Errors occurred.

CAVEATS

The full set of variables is undocumented and should probably be pared down.

SEE ALSO

remote(5n), phones(5n).

NAME

title — title a vector or a GPS

SYNOPSIS

title [**—b**] [**—c**] [**—l***string*] [**—u***string*] [**—v***string*] [*file ...*]

DESCRIPTION

Title prepends *title* to a vector or appends *title* to a GPS. Input is taken from *file(s)* if given, else from the standard input.

If there are any spaces in a title *string*, each space must be escaped (preceded by a backslash).

OPTIONS

—b Make the GPS *title* bold.

—c Retain lower case letters in *title*, otherwise all letters are upper case.

—l*string*
For a GPS, generate a lower *title*: = *string*.

—u*string*
For a GPS, generate an upper *title*: = *string*.

—v*string*
For a vector, *title*: = *string*.

EXAMPLES

Assuming A is a vector, the following example prepends the title 'Vector A' to A.

```
title -c,v"Vector\ A" A
```

Assuming A.g is a GPS, the following example appends 'LOWER' as a lower title and 'UPPER' as an upper title to A.g.

```
title -lLower,uUpper A.g
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

total — sum total

SYNOPSIS

total [*vector* ...]

DESCRIPTION

Output is the sum total of the elements in the input *vector(s)*. If no *vector* is given, the standard input is assumed.

EXAMPLES

The following example outputs the sum total of the elements in A.

```
total A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

NAME

touch — update last modification date of a file

SYNOPSIS

touch [**—c**] [**—f**] [*filename . . .*]

DESCRIPTION

Touch attempts to set the modified date of each *filename*. If a *filename* exists, this is done by reading the first character from the file and writing it back. If a *filename* does not exist, an attempt will be made to create it unless the **—c** option is specified. The **—f** option will attempt to force the touch in spite of read and write permissions on a *filename*.

OPTIONS

—c Unless **—c** is specified, an attempt will be made to create *filename* if it does not exist.

—f An attempt is made to force the touch in spite of read and write permissions on a *filename*.

EXAMPLES

The following example with update the last modification date of all files in the directory */tmp*:

```
touch /tmp/*
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

utimes(2).

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*. When *string2* is short it is padded to the length of *string1* by duplicating its last character. Any combination of the options **—cds** may be used.

In either string, the notation *a—e* means a range of characters from *a* to *e* in increasing ASCII order. The backslash character (\) followed by one, two, or three octal digits stands for the character whose ASCII code is given by those digits. A backslash (\) followed by any other character stands for that character.

OPTIONS

—c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal.

—d Deletes all input characters in *string1*.

—s Squeezes all strings of repeated output characters that are in *string2* to single characters.

EXAMPLES

The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of letters. The second string is quoted to protect the backslash (\) from the Shell. 012 is the ASCII code for newline.

```
tr -cs A-Za-z '\012' <file1 >file2
```

RETURN VALUE

[NO_ERRS] Command completed without error.

[USAGE] Incorrect command line syntax. Execution terminated.

CAVEATS

Will not handle ASCII NUL in *string1* or *string2;* always deletes NUL from input.

Characters in a string that do not appear to be a legitimate range (*a-e*) will be interpreted literally. For example, *e-a* will be interpreted as a string composed of the characters *e*, –, and *a*.

SEE ALSO

ed(1), expand(1), ascii(7).

NAME

trap — execute command on a given signal (**sh** built–in)

SYNOPSIS

trap [*arg*] [*n*]

DESCRIPTION

Arg is a command to be read and executed when the shell receives the signal(s) *n*. (Note that *arg* ss scanned once when the trap is set and once when the trap is taken.) **Trap** commands are executed in order of signal number. If *arg* is absent, all trap(s) *n* are reset to their original values. If *arg* is the null string or nonexistent, this signal is ignored by the shell and by invoked commands. If *n* is 0, the command *arg* is executed on exit from the shell; otherwise upon receipt of signal *n* as numbered in *signal(3c)*. **Trap** with no arguments prints a list of commands associated with each signal number.

EXAMPLES

The following command causes interrupts (signal number 2) to be ignored:

```
trap '' 2
```

The following shell script shows a major use for the **trap** command. The script puts a copy of the standard input into a temporary file and then copies the data to the standard output preceded by the current date. If the interrupt or hangup signal (signal number 1) is sent to the process, the temporary file is removed and the exit code is 1. When the process is finished, the temporary file is removed and the exit code is 1.

```
#!/bin/sh
Excode=1
trap 'rm -f /tmp/slowpr.$$;exit $Excode' 0 1 2
while read f
do
        echo "$f" >> /tmp/slowpr.$$
done
date
cat /tmp/slowpr.$$
Excode=0
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	One or more of the signal numbers is invalid.

CAVEATS

It is very important to note that command and variable substitutions are made when the **trap** command is interpreted *and also* when the corresponding commands are executed. Take, for example, the following commands:

```
trap "rm -f `ls -d /tmp/foo.*`" 0
trap 'rm -f `ls -d /tmp/foo.*`" 0
```

The first command tells the shell to remove all files that matched the pattern */tmp/foo.** at the time the **trap** command was interpreted. The second **trap** command tells the shell to remove all files that matched pattern at the time the shell script exits. This means that in the first case, new files created by the shell script after the **trap** command was executed will not be removed.

Signal number 11 (segmentation violation) can not be trapped at this time. Attempts to do so result in an error.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), onintr(1csh), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2), signal(3c).

NAME

true, false — provide truth values

SYNOPSIS

true

false

DESCRIPTION

True and **false** are usually used in a Bourne shell script. They return the appropriate status "true" or "false" and are usually used in "while" and "until" loops.

These commands are also built-in *sh* commands for speed.

EXAMPLES

```
while true
do
command list
done
```

RETURN VALUE

[0]	The equivalent of true in **sh** conditional tests.
[1]	The equivalent of false in **sh** conditional tests.

SEE ALSO

csh(1csh), sh(1sh), true(1sh).

NAME

true, false — provide truth values (**sh** built–in)

SYNOPSIS

true

false

DESCRIPTION

True and **false** return appropriate status "true" or "false" for use in shell scripts. They are usually used in "while" and "until" loops.

These commands are also available as regular commands for use in other shells and programs.

EXAMPLES

```
while true
do
command list
done
```

RETURN VALUE

[0]	The equivalent of true in **sh** conditional tests.
[1]	The equivalent of false in **sh** conditional tests.

SEE ALSO

csh(1csh), sh(1sh), true(1).

NAME

tset, reset — set up terminal modes and environment

SYNOPSIS

tset [—] [**—A**] [**—E**[*c*]] [**—I**] [**—P**] [**—Q**] [**—S**] [**—a***type*]
[**—d***type*] [**—e**[*c*]] [**—h**] [**—k**[*c*]]
[**—m** [*identifier*][*speed-expression*]:[[**?**]*type*]] [**—n**] [**—p***type*]
[**—r**] [**—s**] [**—t**]
reset *tset options*

DESCRIPTION

Tset and **reset** are used to set up the terminal and the environment. **Tset** is usually used in the shell startup file ($HOME/.profile for users of *sh(1sh)* and $HOME/.login for users of *csh(1csh))* to set up the terminal at login. **Reset** is used to put the terminal back into a "sane" state, usually after abnormal termination of a command that has changed the terminal port modes, or after a line problem has messed up the terminal itself.

First, the terminal type is obtained. There are a number of steps taken to do this. First, the value of the environment variable TERM is obtained. If this is not set, the file */etc/ttytype* is read to find the default terminal type for the port. At this point, the **—m** option(s) are processed if given (See **Mapping** below).

Using the terminal type, the file */etc/termcap* (or the file named by the value of the environment variable TERMCAP) is read to find the termcap entry for the terminal type.

Next, the terminal modes, such as the erase and kill characters and the proper tty driver, are set. In addition, the initialization or reset strings are printed. If the command is invoked as **tset,** the value of the termcap entry "is", or the contents of the file named by the value of the entry "if" is printed. If invoked as **reset,** the entries "rs" and "rf" are used instead.

Parity Setting

If the termcap entry for the terminal contains the flag entry "EP", even parity is set. If "OP" is present, odd parity is set. If neither of these are given, both odd and even parity are set, unless the flag entry "NP" (no parity) is present. If the **—P** option is given, none of this processing takes place, thus the current parity setting is retained.

Mapping

The **—m** option is used when the value of the TERM variable or information in */etc/ttytype* may not be correct. This is useful with terminal types such as "dialup", "network", "arpanet", or "plugboard", whenever the port may be shared by users with different terminals, or when the terminal has more than one possible setup or variation.

The syntax of the **—m** option is
—m [*identifier*][*speed-expression*]:[[**?**]*type*]
The *identifier* is a terminal type string. The *speed-expression* is an expression that involves a combination of the relational operators '='

(equality), '@' (equality), '<' (less than), '>' (greater than), and '!' (not), and a baudrate number (optionally preceded by the character 'B'). The *type* is the name of the terminal type to map to. The default for *type* is "unknown".

The meaning of the **—m** option is that if the currently–known terminal type is the same as *identifier* (which defaults to any terminal type) and the *speed-expression* is true for the currently–known baudrate, the *type* given is to be used instead of the currently–known type. If the *type* string begins with a '?', the user is asked if the terminal type is the correct one to use. The user may then type a carriage return to accept the value printed, or may enter the terminal type desired. The **—A** option also accomplishes this.

Here are a few examples of the **—m** option.

```
-m dialup@1200:4105
```

This mapping says that if the terminal type is "dialup" and the baudrate is 1200, the terminal type to be used is 4105.

```
-m network:?aaa-48
```

This mapping says that if the terminal type is "network" the user is to be asked if the terminal type should be "aaa-48".

```
-m 4107-s-vb<1200:?4107
```

This mapping says that if the terminal type is "4107–s–vb" and the baudrate is less than 1200, the user is to be asked if the terminal type should be 4107 (in this case, it may be that the user might not want to use the status line and visual bells at such a slow baudrate).

Environment Variable Setting

One of the most useful features of **tset** is to set up the environment variables TERM and possibly TERMCAP. The simplest way to use **tset** to set up the environment is to use the option **—**, which prints the terminal type. For example, a user of *sh(1sh)* might use the command line

```
export TERM; TERM=`tset - (options)`
```

where (options) would be any mappings or other options (but not **—s** or **—S).** This sets the variable TERM to the value used by **tset** to set up the terminal.

Since */etc/termcap* contains a lot of entries, it is often desirable to have the environment variable TERMCAP contain the actual terminal capability entry instead of just the name of the file that contains the entry. By having the entry in the environment, programs that use this information, such as *vi(1)* and *more(1),* spend much less time locating this information. This can be done by using the **—s** option (and **—S** in some cases). This option causes commands to be printed which would set the environment variables TERM and TERMCAP. The content of the commands is determined by the value of the environment variable SHELL. If the value ends with "csh", the commands printed will look like:

```
set noglob;
setenv TERM '(type)';
setenv TERMCAP '(termcap entry)';
unset noglob;
```

Otherwise, the shell is assumed to be *sh(1sh)* and the commands printed will look like:

```
TERM='(type)';
TERMCAP='(termcap entry)';
export TERM TERMCAP;
```

When *csh(1csh)* is being used, it is also possible to use the **—S** option. This option only prints the values of the TERM and TERMCAP variables; no commands.

See the EXAMPLES section for sample uses of the **—s** and **—S** options.

OPTIONS

— Report the terminal type, checking to make sure it is valid. No initialization is done.

—A
Ask the user if the terminal type being used is the one desired.

—E[*c*]
Set the erase character to *c* unless terminal can not backspace (like model 33 Teletype). If no *c* is given, ^H (backspace) is used.

—I Do not initialize. No initialization sequence ("is" in the termcap entry) or reset sequence ("rs" in the termcap entry) is printed.

—P
Leave parity alone. See **Parity Setting** above.

—Q
Do not print the value of the erase and kill characters.

—S
Print the terminal name and termcap entry text.

—a*type*
Equivalent to **—m arpanet:***type*.

—d*type*
Equivalent to **—m dialup:***type*.

—e[*c*]
Set the erase character to *c*. If no *c* is given, ^H (backspace) is used.

—h Don't get the terminal type from the environment or the file */etc/ttytype* when doing mapping.

—k[*c*]
Set the kill (erase line) character to *c*. If no *c* is given, ^X is used.

—m [*identifier*][*speed-expression*]**:**[[**?**]*type*]
Map terminal types and baudrates to other terminal types. More than one **—m** option may be given. See the DESCRIPTION section for more information.

—n Use the new tty driver if it is available. This is useful for users of *sh(1sh)* that wish to use some of the new tty features.

—p*type*
Equivalent to **—m plugboard:***type*.

—r Report the terminal name. This is similar to the — option, except that initialization is done.

—s Print commands to set the environment variables TERM and TERMCAP. The value of the environment variable SHELL is used to determine the commands to print.

—t Do not re–use the TERMCAP variable. This forces */etc/termcap* to be reread if it is to be used.

EXAMPLES

The three following examples assume a user that usually uses a Tektronix 4107 when logged in over a dialup port at 1200 baud, and a Tektronix 4105 when logged in over a dialup port at 300 baud. The examples also assume that the terminal type found in the variable TERM or in */etc/ttytype* is valid in all other cases.

This example shows a command line that might be found in the file $HOME/.profile (run by *sh(1sh)* at login time). The result is that the variables TERM and TERMCAP are set to contain the name of the terminal and the value of the termcap entry, respectively. The initialization string and the final values of the erase and kill characters are printed.

```
eval `tset -s -mdialup@1200:?4107 -mdialup@300:?4105`
```

This example shows a set of command lines that might be found in the file $HOME/.login (run by *csh(1csh)* at login time). The result is the same as in the previous example, except that the values of the erase and kill characters are not printed.

```
set noglob
eval `tset -s -Q -mdialup@1200:?4107 -mdialup@300:?4105`
unset noglob
```

Note that the setting of the "noglob" variable is required, due to the way that *csh(1csh)* interprets command lines. The unsetting of the "noglob" variable is not required, but is useful when **tset** is interrupted.

This example is an alternative method of setting TERM and TERMCAP for *csh(1csh)* users. The result is the same as in the previous example, but the option —**S** is used instead of —**s.**

```
set noglob
set terminfo=(`tset -S -Q -mdialup@1200:?4107 -mdialup@300:?41(
setenv TERM "$terminfo[1]"
setenv TERMCAP "$terminfo[2]"
unset noglob
unset terminfo
```

This example resets the terminal modes and prints the "reset terminal" string (using the termcap entry "rs" or "rf").

```
reset
```

FILES

/etc/ttytype	Terminal port information.
/etc/termcap	Default terminal capability database.

VARIABLES

TERM	The user's terminal type.
TERMCAP	The name of the terminal capability database or the capability entry for the terminal itself.
SHELL	The shell being used.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.

CAVEATS

When the value of the environment variable TERMCAP contains something other than a filename (a filename being any string beginning with the character '/'), and the **—t** option is not given, the TERMCAP string is checked to see if it a termcap entry for the value in TERM. If it is, that string is used (this means that **tset** and **reset** run very fast after TERMCAP is set). If not, */etc/termcap* is used as the terminal capability database. This means that a user using a personal termcap file and a terminal type not found in */etc/termcap* may not get the expected output from **tset** or **reset.**

When invoked as **reset,** no error messages are printed when the terminal type can not be located in the termcap file.

The data given with the **—m** option is not checked for syntax, so an invalid use of this option may give strange results.

SEE ALSO

csh(1csh), more(1), sh(1sh), stty(1), vi(1), tty(4), termcap(5t).

NAME

ttoc — make textual table of contents

SYNOPSIS

ttoc [*file* ...]

DESCRIPTION

Output is the textual table of contents (TTOC) generated by the .TC macro of *mm(1)*. The input is assumed to be a *mm* file that uses the .H family of macros for section headers. If no *file* is given, the standard input is assumed.

FILES

*/usr/lib/graf/ttoc.d/** Scripts for *ed(1)* used by **ttoc**.

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mm(1), mod(1g), nroff(1), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), var(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g)*.

NAME

tty — get terminal name

SYNOPSIS

tty [**—s**]

DESCRIPTION

Tty prints the pathname of the user's terminal (found by checking the standard input) unless the **—s** (silent) option is given.

OPTIONS

—s Silent. Suppresses output.

EXAMPLES

The following example shows how **tty** can be used in a shell script to see if the shell script is being run with standard input as a terminal:

```
#!/bin/sh
tty -s
if test $? -eq 1
then
        echo "Input is not from a terminal."
else
        echo "Input is from a terminal."
fi
```

RETURN VALUE

[0]	Standard input is a terminal.
[1]	Standard input is not a terminal.

CAVEATS

When using **tty** in a shell script, it is important to note that if the standard input is being redirected, as in a pipe (¦), the terminal name cannot be determined.

SEE ALSO

test(1sh), ttyname(3c).

NAME

type, which — report command interpretation (*sh* built–in)

SYNOPSIS

type [*name*]

DESCRIPTION

Each *name* is described in terms of what would be executed when that command is executed.

Builtin commands are described by a line like : `name is a shell builtin`.

Functions are described by a line like : `name is a function`, followed by the text of the function.

Hashed commands are described by a line like: `name is hashed (pathname)`, where *pathname* is the current known location of the command.

Regular commands are described by a line like : `name is pathname`, where *pathname* is the pathname of the command as found by the execution path.

Commands that do not exist are described by a line like : `name not found`.

The command **which** is a synonym for **type**.

EXAMPLES

To find out what would be executed by typing **ls**, execute the command

```
type ls
```

The command

```
type trap
```

always prints `trap is a shell builtin`.

RETURN VALUE

[NO_ERRS] Command completed without error.

CAVEATS

The commands **type** and *pathof(1)* are similar, but not the same. **Pathof** only knows how to search the execution path and will only print filenames, whereas **type** knows about builtin commands and functions.

SEE ALSO

break(1sh), cd(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pathof(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), execve(2).

NAME

ul — do underlining

SYNOPSIS

ul [**—i**] [**—{tT}** *terminal*] [*filename . . .*]

DESCRIPTION

Ul reads the named files (or standard input if none are given) and translates occurrences of underscores, and other sequences of specially highlighted text, to the sequence which indicates the special mode for the terminal in use, as specified by the environment variable **TERM**.

When the **—i** option is not specified, the highlighting characters are taken from the termcap entry. The termcap entries that may be used are **mh**, which indicates dim mode, **md**, indicating bold mode, **mr**, indicating reverse video mode, and **me**, which turns the other three off. If these are not given, they default to the contents of the **so** entry. Reverse video is used for the alternate character set.

OPTIONS

—i Causes **ul** to indicate standout modes by printing a separate line under each text line containing an underscore character (_) for underscores, **!** for bold mode, **g** for alternate character set, a caret (^) for superscripts, **v** for subscripts, spaces for normal mode, and *X* for other modes. This is useful when you want to look at the underlining which is present in an **nroff** output stream on a CRT–terminal.

—t *terminal*
Overrides the terminal type specified in the environment. The termcap entry (found in */etc/termcap* by default) is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, **ul** works exactly like *cat(1)*. If the terminal cannot underline, underlining is ignored.

—T *terminal*
The same as **–t**.

EXAMPLES

The following example runs the contents of the file *example.nr* through *nroff(1),* filters the formatted text through **ul**, printing it for display on a Tektronix 4105 terminal:

```
nroff example.nr ¦ ul -t4105
```

FILES

/etc/termcap Default terminal capability file.

VARIABLES

TERM The type of the terminal being used.

TERMCAP	The name of the terminal capability file, or the actual entry for the terminal.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

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.

SEE ALSO

cat(1), col(1), more(1), nroff(1), page(1), termcap(5t).

NAME

ulimit — impose file size limit (**sh** built–in)

SYNOPSIS

ulimit [**—f**] [*n*]

DESCRIPTION

The maximum size (in 512–byte blocks) is changed to *n*. If no value is given, the current limit is printed.

This only limits the size of a file being written to.

OPTIONS

—f Imposes file size limit. This is the default.

EXAMPLES

Executing the command

```
ulimit -f 100
```

causes an error to occur whenever a file being written to becomes larger than 51200 bytes.

RETURN VALUE

[NO_ERRS] Command completed without error.

CAVEATS

Since this system supports a greater variety of limits, there should be a way to set them as well.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), limit(1csh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), unlimit(1csh), umask(1sh), unset(1sh), wait(1sh), which(1sh), execve(2), getrlimit(2), setrlimit(2).

NAME

umask — set user's file creation mask (*csh* built–in)

SYNOPSIS

umask [*nnn*]

DESCRIPTION

The user file creation mask is set to the octal value *nnn*. If *nnn* is omitted, the current value of the mask is printed.

The mask is basically the complement of the file mode. The bits set in the mask will not be set when a file is created or in some types of mode changes (see *umask(2)* and *chmod(1)).*

EXAMPLES

The following command sets the user's file creation mask to 002, which means that files are created with mode 664, meaning the owner and owner's group have read and write permission and others have only read permission.

```
umask 2
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the message occurred.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), chmod(1), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1sh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), umask(2).

NAME

umask — set user's file creation mask (*sh* built-in)

SYNOPSIS

umask [*nnn*]

DESCRIPTION

The user file creation mask is set to the octal value *nnn*. If *nnn* is omitted, the current value of the mask is printed.

The mask is basically the complement of the file mode. The bits set in the mask will not be set when a file is created or in some types of mode changes (see *umask(2)* and *chmod(1)).*

EXAMPLES

The following command sets the user's file creation mask to 002, which means that files are created with mode 664, meaning the owner and owner's group have read and write permission and others have only read permission.

```
umask 2
```

RETURN VALUE

[NO_ERRS] Command completed without error.

CAVEATS

Invalid arguments are not ignored or reported. For example, the command **umask** *filename* will cause the file creation mask to be set to 0.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), chmod(1), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), unset(1sh), wait(1sh), which(1sh), chmod(2), execve(2), umask(2).

NAME

alias, unalias — alias substitutions (**csh** built-in)

SYNOPSIS

alias [*name* [*wordlist*]]
unalias *pattern*

DESCRIPTION

The shell maintains a list of aliases which can be established, displayed, and modified by the **alias** and **unalias** commands. With no arguments, **alias** prints the names and wordlists for all aliases set. With a *name,* the wordlist for that name is printed. With a *name* and a *wordlist,* the wordlist is subjected to command and filename substitution and assigned to the name as an alias. The **unalias** command deletes any aliases that match the given *pattern.* The command **unalias** * deletes all aliases.

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 you can **alias print 'pr \!* ¦ lpr'** to make a command which **pr**'s its arguments to the line printer.

EXAMPLES

The following command causes the command "ls -lR" to be substituted for occurrences of the word "list" when it occurs in a command position.

```
alias list ls -lR
```

This command will delete any two-character aliases that end with the letter 'e'.

```
unalias ?e
```

CAVEATS

Aliases can not contain the names "alias" or "unalias".

If an alias contains it's own name in backquotes, an infinite loop may occur, which will crash the shell. An example of one of these is

```
alias ls 'echo `ls`'
```

Aliases can not always be used in place of simple commands, such as in simple **if** statements.

When one command of a multi–command alias is suspended, the other commands are forgotten by the shell. If all commands of a multi–command alias need to be executed without being affected by suspension, the alias should be surrounded by parentheses, as in the following.

```
alias change '(co -l \!* ; vi \!* ; ci -u \!* )'
```

SEE ALSO

@(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

uname — print name of current system

SYNOPSIS

uname [**—A**] [**—P**] [**—a**] [**—m**] [**—n**] [**—p**] [**—r**] [**—s**] [**—v**]

DESCRIPTION

Uname prints the current system name on standard output. It is mainly useful to determine which system one is using. The options cause selected information returned by *uname(2)* to be printed.

OPTIONS

—A Print all the information for all options.

—P Print all the information for the software packages installed.

—a Print all the information for options **–m, –n, –r, –s,** and **–v.**

—m Print the machine hardware name.

—n Print the nodename (the nodename is set with *sethostname(2)*).

—p Print the name(s) of the software packages installed.

—r Print the operating system release.

—s Print the system name (default).

—v Print the operating system version.

EXAMPLES

The following example prints the machine name.

```
$ uname -m
Tek6130
```

FILES

/usr/lib/bom directory where software packaging info is.

SEE ALSO

bom(5), sethostname(2), uname(2).

NAME

uncompact — uncompress files

SYNOPSIS

uncompact [*name*...]

DESCRIPTION

Uncompact restores the original file from a file compressed by **compact**. If no filenames are given, the standard input is uncompacted to the standard output.

EXAMPLES

This example uncompacts the data in the file *prog.c.C*, leaving the uncompacted data in the file *prog.c* and removing *prog.c.C*:

```
uncompact prog.c.C
```

FILES

**.C*	Compacted file created by **compact**, is removed by **uncompact**.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

ccat(1), compact(1).

NAME

unexpand — change spaces to tabs

SYNOPSIS

unexpand [**—a**] [*filename* ...]

DESCRIPTION

Unexpand puts tabs back into the data from the standard input or the named files and writes the result on the standard output. By default, only leading blanks and tabs are reconverted to maximal strings of tabs.

OPTIONS

—a Tabs are inserted whenever they would compress the resultant file by replacing two or more characters.

EXAMPLES

The following example copies the data in the file *text* to the file *text.t*, replacing spaces with tabs wherever they would save space:

```
unexpand -a text > text.t
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

expand(1).

NAME

unget, sact — undo a previous get of an SCCS file

SYNOPSIS

unget [—r*SID***] [—s] [—n]** *filenames*

DESCRIPTION

Unget undoes the effect of a **get —e** command done prior to creating the intended new delta. If a directory is named, **unget** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of — (a dash) 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.

OPTIONS

—n Causes the retention of the gotten file which would normally be removed from the current directory.

—r*SID*
Uniquely identifies which delta is no longer intended. (This would have been specified by **get** as the *new delta*). The use of this keyletter is necessary only if two or more outstanding **get**s for editing on the same SCCS file were done by the same person (loginname). 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*.

DIAGNOSTICS

Use *sccshelp(1sccs)* for explanations.

SEE ALSO

delta(1sccs), get(1sccs), sccshelp(1sccs), sccsfile(5sccs).

NAME

uniq — report repeated lines in a file

SYNOPSIS

uniq —{**u,d,c**} [+*n*] [—*n*] [*input*[*output*]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. The second and succeeding copies of repeated lines are removed; the remainder is written on the output file. 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 default output is the union of the —**u** and —**d** mode outputs.

The —**c** option generates an output report in default style but with each line preceded by a count of the number of times it occurred.

Only one of the flags **u** , **d** , or **c** can be used at a time.

The *n* arguments specify skipping an initial portion of each line in the comparison.

OPTIONS

—**c** Generates a default output with each line preceded with a count of how many times it occurred.

—**d** Only the first repeated line is output.

—*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of nonspace, nontab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

—**u** Repeated lines in the original file are not output.

EXAMPLES

Typically this command is used on previously sorted files like *sorted*. In the following example the output is written on a new file *uniqed*:

```
uniq sorted uniqed
```

All lines in the file *uniqed* will be distinct, since the input to **uniq** was previously sorted.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

Lines over 1024 characters in length will be truncated.

SEE ALSO

awk(1), comm(1), cut(1), egrep(1), fgrep(1), grep(1), join(1) look(1), paste(1), sort(1).

NAME

limit, unlimit — set resource limitations (**csh** built-in)

SYNOPSIS

limit [*resource* [*maximum*]]
unlimit [*resource*...]

DESCRIPTION

With no arguments, **limit** prints all resource limits (see below for a list of limitable resources). With only a *resource,* the limit for that resource is printed. With a *resource* and a *maximum,* the limit for that resource is set to the given maximum, if possible. This limit will apply to the current shell and is inherited by all child processes.

The **unlimit** command removes the limit on the named resources. If no arguments are given, all resource limits are removed.

Resources that are currently controlable include *cputime* (the maximum number of CPU–seconds to be used by each process), *filesize* (the largest single file which can be created), *datasize* (the maximum growth of the data + stack region via *sbrk(2)* beyond the end of the program text), *stacksize* (the maximum size of the automatically–extended stack region), and *coredumpsize* (the size of the largest core dump that will be created).

The *maximum-use* may be given as a (floating point or integer) number followed by a scale factor. For all limits other than *cputime* the default scale is **k** or *kilobytes* (1024 bytes); a scale factor of **m** or *megabytes* may also be used. For *cputime* the default scaling is *seconds*, while *m* for minutes or *h* for *hours*, or a time of the form *mm:ss* giving minutes and seconds may be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

EXAMPLES

This example limits the size of all 'core' files generated to 0 bytes. This means that no 'core' files will be created if a program terminates abnormally.

```
limit coredumpsize 0
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type given in the message occurred.

CAVEATS

A child process may unlimit any resources limited by the parent.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh),

jobs(1csh), kill(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), ulimit(1sh), umask(1csh), unhash(1csh), unalias(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), brk(2), getrlimit(2), sbrk(2), setrlimit(2), ulimit(3c).

NAME

set, unset — set and unset shell variables (**csh** built-in)

SYNOPSIS

set [*expression...*]
unset *pattern*

DESCRIPTION

The **set** command is used to set and change the values of *csh(1csh)* variables. Shell variables can either be single words or vectors (arrays) of words. If no arguments are given, the names and values of all shell variables are printed, with vectors surrounded by parentheses. The *expression* arguments may have the following forms:

name — The named variable is set to the null string.

name = *word* — The named variable is set to the word given. The word is subject to substitutions, so special characters must be escaped or quoted.

name [*index*] = *word*
The *index*'th component of the vector *name* is set to the value of *word*. The component must already exist.

name = *(wordlist)*
The named vector is set to the list of values in the *wordlist*.

In all cases, variable expansion is done before any assignment, so the command sequence

```
set x=hello
set x=goodbye y=(you say $x)
```

will set the variable x to "goodbye" and the vector y to "(you say hello)", not "(you say goodbye)".

The **unset** command is used to delete variables that match the given *pattern*. The command **unset** * unsets all variables.

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

Though shell variables are not environment variables (see *setenv(1csh)*), the variables *path,* and *cdpath,* are imported by the shell, and the values of *home, term, user, path,* and *cdpath* are exported to the environment. The shell variable *path* and the environment variable PATH are always maintained together, whereas *cdpath* and CDPATH are maintained together but the values may be changed (see *cd(1sh)* and *cd(1csh)* for more information).

Following is a description of variables which affect built–in commands:

argv Set to the arguments to the shell, it is from this variable that positional parameters are substituted; for example, $1 is replaced by **$***argv[1]*, and so forth.

cdpath Gives a list of alternate directories searched to find subdirectories in **chdir** commands. This variable is maintained along with the CDPATH environment variable.

complete When set, this variable enables command completion in interactive shells.

cwd The full pathname of the current directory. See CAVEATS.

dirname Set to the default directory to change to when the argument to *cd(1csh)* or *pushd(1csh)* is not found in the current directory or in the list of directories found in the *cdpath* variable.

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. In all cases, redirection is not taken into account, so the commands

```
set echo
cat /etc/group )& foo
```

will result in echoing "cat /etc/group" to the original standard error, instead of into the file "foo".

hardpaths When symbolic links are followed when changing directories, the value of the variable *cwd* and the values on the directory stack may not be proper pathnames. When this variable is set, each change of directory causes the current directory to be expanded to not include symbolic links. This should only be used when accuracy is required, as it makes directory changes much slower.

histchars Can be given a string value to change the characters used in history substitution. The first character of its value is used as the history substitution character, replacing the default character **!**. The second character of its value replaces the character ↑ in quick substitutions.

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 the tilde (~) 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 ⟨CTRL–D⟩'s.

list When set to nothing, this variable enables file and command listing using the ^D character in interactive mode. If set to 'f', filename marking is turned on. If set to 'l', filename marking is turned on and symbolic links are specially marked.

listpathnum This modifies the output of file listing (see the *list* variable) so that the number of the path element where each of the commands is found is printed.

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 filename* when there is mail in *filename.*

If this variable is not set, no checking for mail is done.

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; for example, **echo [** still gives an error.

notify If set, the shell notifies asynchronously of job completions. The default is to gather present job completions just before printing a prompt.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable then only full pathnames will execute. The usual search path is ., */bin*, and */usr/bin*, but this may vary from system to system. For the superuser, 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 superuser.

savehist is given a numeric value to control the number of entries of the history list that are saved in *~/.history* when the user logs out. Any command which has been referenced in this many events will be saved. During start up, the shell sources ~/.history into the history list enabling history to be saved across logins. Too large values of **savehist** will slow down the shell during start up.

shell The file in which the shell resides.

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

time Controls automatic timing of commands and time summary format. If the first value in the vector is set, then any command which takes more than this many CPU seconds will cause a time and resource usage summary to be printed when it terminates. If the second value in the vector is set, that value controls the summary format. See *time(1csh)* for the summary format values and the default format.

vbell If set to nothing, command completion errors will be signaled by the visible bell (:vb:) sequence from the termcap entry (see *termcap(5t)*) instead of the terminal bell. If set to anything else, the value of the variable is printed.

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

EXAMPLES

The following command line sets the value of the variable 'Here' to the current working directory, the vector 'files' to the names of the files in the current directory which begin with the letter 't', and the fourth element of the variable 'junk' to the word "foo". If 'junk' does not already exist and have at least four elements, an error message will result.

```
set Here="`pwd`" files=( t* ) junk[4]=foo
```

This example unsets the values of all single–letter variables.

```
unset ?
```

DIAGNOSTICS

name: undefined variable.
An element of the variable *name* was assigned to, but the name has never been assigned.

set: subscript out of range.
An element of a variable was assigned to, but that element does not exist.

CAVEATS

The value of the variable *cwd* is not always correct, especially when symbolic links are followed. If this variable is required to be correct, the variable *hardpaths* should be set.

When command completion or file listing are turned on, terminal echo mode will always be turned on and the terminal will always be set to cooked input mode. This is done to ensure that command completion and file listing behave properly. If you need to change these terminal characteristics, you must turn off command completion and file listing.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1sh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1sh), unsetenv(1csh), wait(1csh), which(1csh).

NAME

unset — remove variables and functions (**sh** built–in)

SYNOPSIS

unset [*name* . . .]

DESCRIPTION

For each *name,* the corresponding variable or function is removed.

The variables **PATH, PS1, PS2, MAILCHECK,** and **IFS** cannot be **unset.**

EXAMPLES

The following command makes sure that there are no functions called **ls** defined:

```
unset ls
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

The **unset** command is NOT the opposite of the **set** command.

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unalias(1csh), unset(1csh), wait(1sh), which(1sh), execve(2).

NAME

setenv, unsetenv — change environment variables (**csh** built–in)

SYNOPSIS

setenv *name value*
unsetenv *pattern*

DESCRIPTION

Environment variables, unlike shell variables, are passed to all child processes in the "environment" (see *environ(7)).*

The command **setenv** sets the environment variable *name* to the given *value,* which must be a single word.

The command **unsetenv** removes all environment variables whose name matches *pattern,* which may contain any metacharacters normally used in filename expansion. This deletes the names from the environment for all child processes, leaving the parent processes unaffected.

Setting the variable PATH results in the shell variable *path* being set similarly. See *set(1csh)* for more information.

EXAMPLES

The following command sets the environment variable PATH to the value ':/bin:/etc:/usr/bin'. Note that this causes the shell variable *path* to be set to '(. /bin /etc /usr/bin)'.

```
setenv PATH :/bin:/etc:/usr/bin
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the message occurred.

CAVEATS

Shell variables take precedence over environment variables. This means that setting the environment variable 'prompt' to 'foo' does not change the prompt for the current or subsequent shells, and that the value of '$prompt' is not 'foo' in a subsequent invocation of **csh.**

There is no way for a child to change the environment of the parent.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), export(1sh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), printenv(1), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), set(1sh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1csh), getenv(3c), environ(7).

NAME

unstrip — add symbol table and relocation bits to a stripped file

SYNOPSIS

unstrip [**—O** *suffix*] [**—S** *suffix*] [**—o**] [**—v**] *file ...*

DESCRIPTION

Unstrip adds the symbol table and relocation bits to a file that is the output of the assembler and loader that has been stripped. This is useful when debugging a program that has been stripped with **strip.**

To use **unstrip,** you must have a symbol table file for the stripped file; such as produced by **strip —s**.

OPTIONS

—O *suffix*
Create an output file named *filesuffix* that is an unstripped version of *file*. The input file is left stripped.

—S *suffix*
Symbol file table is named *filesuffix*. If you don't use this option, **unstrip** looks for a symbol table file named *file.symtab*.

—o Create an output file named *file.unstripped* that is an unstripped version of *file*. The input file is left stripped.

—v Verbose. Reports the names of all input, output, and symbol table files used.

EXAMPLES

The following invocation adds the symbol table and relocation bits to the file *cmd* from the file *cmd.symtab* and prints messages telling you what it did:

```
unstrip -v cmd
```

The following invocation creates an unstripped version of the file *cmd* in a file named *cmd.out*. The symbol table information is taken from *cmd.symbols*.

```
unstrip -O ".out" -S ".symbols" cmd
```

FILES

/tmp/stm? Temporary file

RETURN VALUE

[NO_ERRS] Command completed without error.

[NP_WARN] An error warranting a warning message occurred. Execution continues.

[P_WARN] A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

[P_ERR] A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

SEE ALSO

ld(1), strip(1), a.out(5).

NAME

uptime — show status of hosts on network

SYNOPSIS

uptime [**—l**][**—t**][**—u**][**—U**][**—r**][**—w** *seconds*][*hostname*]...

DESCRIPTION

Uptime with no arguments, prints statistics about the local host. The statistics include the current time, the length of time the system has been up, how many users are currently logged on, and the average number of jobs in the run queue over the last one, five, and fifteen minute time periods.

Uptime can also be used to obtain similar statistics from other hosts on the network. If any of the options are specified but no hostname is given, **uptime** will broadcast for uptime statistics on the local area network. Normally uptime will listen for responses for 10 seconds and then display them. If desired, the timeout period can be lengthened or decreased through the **—w** *[seconds]* option.

Hosts which at one time responded to an uptime request, but did not respond to the most recent request, will be marked as "down". A host that is down for more than 5 days will be deleted from the uptime data base (in */usr/spool/uptime*).

If one or more hostnames is specified on the **uptime** command line, uptime will interogate only those hosts for the statistics. Also, by specifying a hostname, it is possible to get uptime statisitics from hosts on other connected networks and not just the local network.

OPTIONS

—l Sort status lines by load average.

—r Generate status lines for remote systems current to the minute.

—t Sort status lines by uptime.

—u Sort status lines by number of users.

—w *[seconds]* When broadcasting for uptime statistics (i.e. when no hostname is specified), the **—w** *[seconds]* option determines how long **uptime** waits collecting responses.

—U Generate a list of host systems that are up, one name to a line. This is useful for shell programs.

FILES

/vmunix	System name list.
*/usr/spool/uptime/up.**	Data files.

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[INTERNAL]	An unexpected error occurred. Execution was terminated. Record the message and save the core file for analysis. Contact service personnel at your Tektronix field office.

CAVEATS

Uptime is incompatible with the standard Berkeley 4.2 offering, **ruptime**.

SEE ALSO

w(1), ruptime(1n), udpd(8n).

NAME

users — who is logged in on local machines or on the current host system

SYNOPSIS

users [**—r**] [**—a**]

DESCRIPTION

Report usernames, separated by spaces, that are logged in to this host. The **—r** switch includes usernames for all machines on the local network. The hosts on a net report information used by *users(1n)* every minute. If a machine has not been heard from for five minutes then it is assumed the machine is down, and *users(1n)* does not report users last known to be logged into that machine.

If a user hasn't typed to a system for an hour or more, then the user will be omitted unless the **—a** flag is given.

This command is merely a link to **who —u**.

OPTIONS

—a Report users on remote systems even if they have been idle an hour or more.

—r Report users on all remote systems that are not idle or have been idle less than one hour.

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[INTERNAL]	An unexpected error occurred. Execution was terminated. Record the message and save the core file for analysis. Contact service personnel at your Tektronix field office.

CAVEATS

Only 40 users can be shown on one host because of network packet size.

SEE ALSO

getuid(2), utmp(5), uptime(1n), rwhod(8n), who(1n).

NAME

uucp — UTek to UTek copy

SYNOPSIS

uucp [**—c**][**—d**][**—m**] *source-filename . . . destination-filename*

DESCRIPTION

Uucp copies files named by the *source-filename* arguments to the *destination-filename* argument. A filename may be a pathname on your machine, or may have the form

```
system-name!pathname
```

where *system-name* is taken from a list of system names which **uucp** knows about. Shell metacharacters ?, *, [, and] appearing in the pathname part will be expanded on the appropriate system.

Pathnames may be one of

(1) a full pathname;

(2) a pathname preceded by ~*user* where *user* is a user ID on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

If the result is an erroneous pathname for the remote system the copy will fail. If the *destination-filename* is a directory, the last part of the *source-filename* 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(1mh)*.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod(2)*).

> **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 pathname; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary pathnames.

OPTIONS

—c Use the source file when copying out rather than copying the file to the spool directory.

—d Make all necessary directories for the file copy.

—m
Send mail to the requester when the copy is complete.

FILES

/usr/spool/uucp	Spool directory.
*/usr/lib/uucp/**	Other data and program files.

RETURN VALUE

[0]	No errors.
[nonzero]	Errors occured.

CAVEATS

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 ?, *, [, and] will not activate the **—m** option.)

SEE ALSO

uux(1n), mail(1mh), uulog(1n).

NAME

uuencode, uudecode — encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [*source*]*remotedest* | **mail** *sys1!sys2!..!decode*
uudecode [*filename*]

DESCRIPTION

Uuencode and **uudecode** are used to send a binary file via **uucp** (or other) mail. This combination can be used over indirect mail links even when *uusend(1n)* is not available.

Uuencode takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the *remotedest* for recreation on the remote system.

Uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user "decode" should be filtered through the **uudecode** program. This way the file is created automatically without human intervention. This is possible on the **uucp** network by either using **sendmail** or by making **rmail** be a link to **Mail** instead of **mail**. In each case, an alias must be created in a master file to get the automatic invocation of **uudecode**.

If these facilities are not available, the file can be sent to a user on the remote machine who can **uudecode** it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

RETURN VALUE

[0]	No Errors
[nonzero]	Errors occured.

CAVEATS

The file is expanded by 35% (three bytes become four, plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking **uudecode** (often **uucp**) must have write permission on the specified file.

SEE ALSO

uuencode(5n), uusend(1n), uucp(1n), uux(1n), mail(1mh).

NAME

uudiff — directory comparison between machines

SYNOPSIS

uudiff [**—d**] *local-name remote-name*

DESCRIPTION

Uudiff compares the files in the directory *local-name* and the directory *remote-name,* (where *remote-name* may be of the form *system-name!directory-name* and *system-name* is a **uucp** UTek name). It reports (via **mail**) which files are added, deleted, or changed, and provides a *diff(1)* of altered printable files.

If a part of *remote-name* is omitted (either the system or the directory) the corresponding part of *local-name* is used. If *local-name* is a file, rather than a directory, *remote-name* is also assumed to be a file and the program degenerates into *diff(1).*

The option **—d** does not **diff** altered files; only the summary by filenames is prepared.

FILES

zz[abcdeglmn]?????	These files are used for scratch space.
name.?????	These files are brought back from the remote machine for **diff**ing.
uudiff.?????	The final report is left here (the exact name is reported by **mail**).

DIAGNOSTICS

There are very few diagnostics given. Anything more serious than misspelling *local-name* causes unpredictable and obscure results.

CAVEATS

In addition to the standard **uucp** requirements, a hook is needed at the remote system, and at present is only installed on the systems *research* and *inter.*

This program is written in Shell and should be translated to C so it could give diagnostics.

Even if *remote-system* is the local system, **uudiff** is subject to delays in **uucp** traffic.

It should probably write the standard output, instead of insisting on going into the background.

Since *checksums* are used, there is a probability of 1 in 2**32 of missing differences between files.

The ~*userid* syntax is not recongnized.

SEE ALSO

diff(1), uucp(1n).

NAME

uuencode, uudecode — encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [*source*]*remotedest* ¦ **mail** *sys1!sys2!..!decode*
uudecode [*filename*]

DESCRIPTION

Uuencode and **uudecode** are used to send a binary file via **uucp** (or other) mail. This combination can be used over indirect mail links even when *uusend(1n)* is not available.

Uuencode takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the *remotedest* for recreation on the remote system.

Uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user "decode" should be filtered through the **uudecode** program. This way the file is created automatically without human intervention. This is possible on the **uucp** network by either using **sendmail** or by making **rmail** be a link to **Mail** instead of **mail**. In each case, an alias must be created in a master file to get the automatic invocation of **uudecode**.

If these facilities are not available, the file can be sent to a user on the remote machine who can **uudecode** it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

RETURN VALUE

[0]	No Errors
[nonzero]	Errors occured.

CAVEATS

The file is expanded by 35% (three bytes become four, plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking **uudecode** (often **uucp**) must have write permission on the specified file.

SEE ALSO

uuencode(5n), uusend(1n), uucp(1n), uux(1n), mail(1mh).

NAME

uulog — maintain a summary log of uucp transactions.

SYNOPSIS

uulog [—**s***sys*] . . . [—**u***user*] . . .

DESCRIPTION

Uulog maintains a summary log of **uucp** and *uux(1n)* transactions in the file */usr/spool/uucp/LOGFILE* by gathering information from partial log files named */usr/spool/uucp/LOG.*.?*. It removes the partial log files.

OPTIONS

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*.

FILES

/usr/spool/uucp	Spool directory.
*/usr/lib/uucp/**	Other data and program files.

RETURN VALUE

[0]	No errors.
[nonzero]	Errors occured.

SEE ALSO

uux(1n), mail(1mh), uucp(1n).

NAME

uuname — list other hosts that local host knows about.

SYNOPSIS

uuname [**—l**]

DESCRIPTION

Uuname lists the **uucp** names of known systems.

OPTIONS

—l Return the local system name.

FILES

/usr/lib/uucp/L.SYS Local system information

RETURN VALUE

[0]	No errors
[nonzero]	Errors occurred.

SEE ALSO

mail(1mh), uucp(1n), uux(1n).

NAME

uusend — send a file to a remote host

SYNOPSIS

uusend [**—m** *mode*] *sourcefilename sys1!sys2!..!remotefilename*

DESCRIPTION

Uusend sends a file to a given location on a remote system. The system need not be directly connected to the local system, but a chain of *uucp(1n)* links needs to connect the two systems.

If the **—m** option is specified, the mode of the file on the remote end will be taken from the octal number given. Otherwise, the mode of the input file will be used.

The sourcefile can be a dash (—), meaning to use the standard input. Both of these options are primarily intended for internal use of **uusend**.

The *remotefilename* can include the ~*userid* syntax.

OPTIONS

—m*mode*
Set the mode of the file on the remote end to *mode*.

— Instead of source file use standard input.

DIAGNOSTICS

If anything goes wrong any further away than the first system down the line, you will never hear about it.

RETURN VALUE

[0]	No errors.
[nonzero]	Errors occurred.

CAVEATS

This command shouldn't exist, since **uucp** should handle it.

All systems along the line must have the **uusend** command available and allow remote execution of it.

Some **uucp** systems have a bug where binary files cannot be the input to a **uux** command. If this bug exists in any system along the line, the file will show up severly munged.

SEE ALSO

uux(1n), uucp(1n), uuencode(1n).

NAME

uux — utek to utek command execution

SYNOPSIS

uux [—] *command-string*

DESCRIPTION

Uux will gather zero or more files from various systems, execute a command on a specified system and send standard output to a file on a specified system.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and filenames may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

Filenames may be one of

(1) a full pathname

(2) a pathname preceded by ~*xxx* where *xxx* is a user ID 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 <, >, ; , and ¦ should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

OPTIONS

— This option will cause the standard input to the **uux** command to be the standard input to the *command-string*.

EXAMPLES

The command

```
uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !fi.diff"
```

will get the *f1* files from the *usg* and *pwba* machines, execute a **diff** command, and put the results in the file *f1.diff* in the local directory.

WARNING

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an incoming request from **uux**. Typically, a restricted site will permit little other than the receipt of **mail** via **uux**.

FILES

/usr/spool/uucp	Spool directory.
*/usr/lib/uucp/**	Other data and programs.

RETURN VALUE

[0]	No errors.
[nonzero]	Errors occurred.

CAVEATS

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.

There is no notification of denial of execution on the remote machine.

SEE ALSO

uucp(1n).

NAME

val — validate SCCS file

SYNOPSIS

val —
val [—**s**] [—**r***SID*] [—**m***name*] [—**y***type*] *filenames*

DESCRIPTION

Val determines if the specified *filename* 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 dash (—), 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 8-bit code returned by **val** is a disjunction of the possible errors; for example, it can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument
bit 1 = unknown or duplicate keyletter argument
bit 2 = corrupted SCCS file
bit 3 = cannot open file or file not SCCS
bit 4 = SID is invalid or ambiguous
bit 5 = SID does not exist
bit 6 = **%Y%**, **—y** mismatch
bit 7 = **%M%**, **—m** mismatch

Note that **val** can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned —— a logical **OR** of the codes generated for each command line and file processed.

OPTIONS

—m*name*

The argument value *name* is compared with the SCCS **%M%** keyword in *filename*.

—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 (for example, **r1** is ambiguous because it physically does not exist but implies 1.1, 1.2, and so forth, which may exist) or invalid (for example, **r1.0** or **r1.1.0** are invalid because neither case can exist as a valid delta number). If the SID is valid and not ambiguous, a check is made to determine if it actually exists.

—**s** 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.

—y*type*
The argument value *type* is compared with the SCCS **%Y%** keyword in *filename*.

DIAGNOSTICS

Use *sccshelp(1sccs)* for explanations.

CAVEATS

Val can process up to 50 files on a single command line.

SEE ALSO

admin(1sccs), delta(1sccs), get(1sccs), prs(1sccs).

NAME

var — variance

SYNOPSIS

var [*vector* ...]

DESCRIPTION

Output is the variance of the elements in the input *vector(s)*. If no *vector* is given, the standard input is assumed.

EXAMPLES

The following example outputs the variance of the elements in A.

```
var A
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), vtoc(1g), whatis(1g), yoo(1g), and *gps(5g).*

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 noncontrol character are copied in their entirety.

A keyword is composed of nine 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 \.

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 (for example,: **012** ⟩ **12** is false). All other operators take strings as arguments (for example,: **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 *message* on the diagnostic output.

:err *message*

Prints the *message* followed by:

```
ERROR: err statement on line ... (915)
```

on the diagnostic output. **Vc** halts execution, and returns an exit code of 1.

OPTIONS

—a Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines, not just in **vc** statements.

—c*char*

Specifies a control character to be used in place of the **:** character.

—s Silences warning messages (not error) that are normally printed on the diagnostic output.

—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.

DIAGNOSTICS

Use *sccshelp (1sccs)* for explanations.

SEE ALSO

sccshelp(1sccs).

NAME

verify — verify mail addresses

SYNOPSIS

verify [*filename*] [**—draft**] [**—remove**] [**—noremove**] [**—verbose**] [**—noverbose**]

DESCRIPTION

Verify is used to expand and verify mail addresses found in the given file or the default draft file.

Verify actually invokes *sendmail(8mh)* with the **-bv** option. For more information about the output, see the manual page for **sendmail.**

The default options to **verify** are: **—remove** and **—noverbose.**

OPTIONS

—draft
If no filename is specified, use the default draft file without asking.

—remove
Do not include the invoker's name in the expansion.

—noremove
Include the invoker's name in the expansion.

—verbose
Invoke **sendmail** with the **—v** option, causing all expansions to be printed.

—noverbose
Terse mode. Alias expansions are not printed.

EXAMPLES

Verify is usually executed via the 'v' command in *comp(1mh), repl(1mh),* and *forw(1mh),* but can be used alone to verify addresses. The following example verifies the addresses in the current draft file, if it exists. No questions will be asked about the file to use, and aliases will be expanded in the output.

```
verify -draft -verbose
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	A problem as reported by the error message occurred.

SEE ALSO

comp(1mh), forw(1mh), repl(1mh), sendmail(8mh).

NAME

ex, e, edit, vi, view — text editor

SYNOPSIS

ex [—] [—**R**] [—**l**] [—**r**] [—**t***tag*] [—**v**] [—**w***n*] [+*command*] name ...
e (**ex** options)
edit (**ex** options)
vi (**ex** options)
view (**ex** options)

DESCRIPTION

Ex is the root of a family of editors: **e, edit, ex, vi,** and **view. Ex** is a superset of **ed,** with the most notable extension being a display editing facility. Display based editing is the focus of **vi.**

If you have 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.**

Executing **vi** is the same as executing **ex** with the —**v** option.

The commands **e** and **edit** are equivalent to *ex* in all respects except that the following options are set: **nomagic** (reduced regular expression syntax), **noopen** (the commands *open* and *visual* are disabled), and **report** is set to 1 (all commands modifying more than 1 line cause a message to be printed).

The command **view** is the same as executing **ex** with the —**v** and —**R** options.

OPTIONS

— Suppresses all interactive–user feedback and is useful in processing editor scripts in command files.

—**R**
Sets the *readonly* option at the start of the editing session.

—**l** Sets up for editing LISP, setting the *showmatch* and *lisp* options.

—**r** Used in recovering after an editor or system crash, retrieving the last saved version of the named file, or if no file is specified, typing a list of saved files.

—**t***tag*
Equivalent to an initial *tag* command, editing the file containing the *tag* and positioning the editor at its definition.

—**v** Visual display editing mode.

—**w***n*
Sets the default window size to *n*.

+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 file initially. Other useful commands here are scanning patterns of the form "/pat" or line numbers, e.g. "+100" starting at line 100.

FILES

/usr/lib/ex?.?strings	Error messages.
/usr/lib/ex3.7recover	Recover command.
/usr/lib/ex3.7preserve	Preserve command.
/etc/termcap	Default file containing terminal capability entries.
~/.exrc	The editor startup file.
/tmp/Exnnnnn	Editor temporary file.
/tmp/Rxnnnnn	Named buffer temporary file.
/usr/preserve	Preservation directory.

VARIABLES

TERM	The type of terminal being used.
TERMCAP	The file containing the terminal capability entry, or the entry itself.
SHELL	The name of the shell program to be used when invoking a subshell or processing special commands.
HOME	The user's home directory.
EXINIT	Specifies an alternate editor startup file or contains the actual startup data.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

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.

If there are more than 300 magic numbers of type long and/or short in the magic number file, no files may be edited.

When metacharacters are used in naming a file (such as with the 'r' command), the user's default shell is used to do the expansion in noninteractive mode. When this shell is **csh,** the *prompt* variable should be checked to see if it is set in the *.cshrc* file. If it isn't, it should not be set, since this will cause **ex** to see more than one filename.

SEE ALSO

awk(1), e(1), ed(1), edit(1), grep(1), egrep(1), sed(1), vi(1), view(1), magic(5), termcap(5t), environ(7).

NAME

ex, e, edit, vi, view — text editor

SYNOPSIS

ex [—] [**—R**] [**—l**] [**—r**] [**—t***tag*] [**—v**] [**—w***n*] [*+command*] name ...
e (**ex** options)
edit (**ex** options)
vi (**ex** options)
view (**ex** options)

DESCRIPTION

Ex is the root of a family of editors: **e, edit, ex, vi,** and **view. Ex** is a superset of **ed,** with the most notable extension being a display editing facility. Display based editing is the focus of **vi.**

If you have 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.**

Executing **vi** is the same as executing **ex** with the **—v** option.

The commands **e** and **edit** are equivalent to *ex* in all respects except that the following options are set: **nomagic** (reduced regular expression syntax), **noopen** (the commands *open* and *visual* are disabled), and **report** is set to 1 (all commands modifying more than 1 line cause a message to be printed).

The command **view** is the same as executing **ex** with the **—v** and **—R** options.

OPTIONS

— Suppresses all interactive-user feedback and is useful in processing editor scripts in command files.

—R
Sets the *readonly* option at the start of the editing session.

—l Sets up for editing LISP, setting the *showmatch* and *lisp* options.

—r Used in recovering after an editor or system crash, retrieving the last saved version of the named file, or if no file is specified, typing a list of saved files.

—t*tag*
Equivalent to an initial *tag* command, editing the file containing the *tag* and positioning the editor at its definition.

—v Visual display editing mode.

—w*n*
Sets the default window size to *n*.

+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 file initially. Other useful commands here are scanning patterns of the form "/pat" or line numbers, e.g. "+100" starting at line 100.

FILES

/usr/lib/ex?.?strings	Error messages.
/usr/lib/ex3.7recover	Recover command.
/usr/lib/ex3.7preserve	Preserve command.
/etc/termcap	Default file containing terminal capability entries.
~/.exrc	The editor startup file.
/tmp/Exnnnnn	Editor temporary file.
/tmp/Rxnnnnn	Named buffer temporary file.
/usr/preserve	Preservation directory.

VARIABLES

TERM	The type of terminal being used.
TERMCAP	The file containing the terminal capability entry, or the entry itself.
SHELL	The name of the shell program to be used when invoking a subshell or processing special commands.
HOME	The user's home directory.
EXINIT	Specifies an alternate editor startup file or contains the actual startup data.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

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.

If there are more than 300 magic numbers of type long and/or short in the magic number file, no files may be edited.

When metacharacters are used in naming a file (such as with the 'r' command), the user's default shell is used to do the expansion in noninteractive mode. When this shell is **csh,** the *prompt* variable should be checked to see if it is set in the *.cshrc* file. If it isn't, it should not be set, since this will cause **ex** to see more than one filename.

SEE ALSO

awk(1), e(1), ed(1), edit(1), grep(1), egrep(1), sed(1), vi(1), view(1), magic(5), termcap(5t), environ(7).

NAME

vihelp — print help information for vi editor commands

SYNOPSIS

vihelp { **add, change, delete, files, help, misc, move, screen, search** }

DESCRIPTION

Vihelp prints help information for vi commands in the category you specify as an argument. You can use **vihelp** from the shell or from the vi editor (see the EXAMPLES section).

EXAMPLES

For information on **vi** commands that add text to a file, type:

```
vihelp add
```

To get the same information while in the **vi** editor, type

```
:!vihelp add
```

FILES

*/usr/lib/help/vi** screen data files

CAVEATS

Vihelp accepts only one argument and no options.

SEE ALSO

vi(1).

NAME

vtoc — visual table of contents

SYNOPSIS

vtoc [**—c**] [**—d**] [**—h***n*] [**—i**] [**—m**] [**—s**] [**—v***n*] [*file* ...]

DESCRIPTION

Vtoc produces a GPS describing a hierarchy chart from a textual table of contents (TTOC). The output drawing consists of boxes containing text, connected in a tree structure. If no *file* is given, the standard input is assumed.

Each TTOC entry describes one box and has the form:

id [*line-weight*,*line-style*] "*text*" [*mark*]

Brackets denote optional fields. Each field describes an attribute of the box, as follows:

id — is an alternating sequence of numbers and periods. The *id* specifies the position of the entry in the tree: the number of periods in the *id* corresponds to the level of the entry in the heirarchy. The *id* is displayed above the upper left corner of the box. The entry with *id* **0.** is the root of the tree.

line-weight — is either:

n, normal–weight; or
m, medium–weight; or
b, bold–weight.

line-style — is either:

so, solid–line;
do, dotted–line;
dd, dot–dash line;
da, dashed–line; or
ld, long–dashed.

text — is a character string surrounded by quotes. The characters between the quotes become the contents of the box. To include a quote within a box, it must be escaped (\").

mark — is a character string (surrounded by quotes if it contains spaces). The *mark* string is put above the top right corner of the box. To include either a quote or a period within a *mark*, it must be escaped (\" or \.).

Entry example: `1.1 b,da "ABC" DEF`

Entries may span more than one line by escaping the newline. For example:

```
2.1.3 n,do \
"text label for box" boxmark
```

Comments are surrounded by the /*,*/ pair. They may appear anywhere in a TTOC.

The command **dtoc** produces a TTOC describing a UTek directory structure. See *dtoc(1g)*.

OPTIONS

—c Take the text as entered (default is all upper case).

—d Connect the boxes with diagonal lines.

—h*n*
Horizontal interbox space is *n*% of box width.

—i Suppress the box *id*.

—m
Suppress the box *mark*.

—s Do not compact boxes horizontially.

—v*n*
Vertical interbox space is *n*% of box height.

EXAMPLES

The following example outputs a VTOC derived from the TTOC in the file A.t. The VTOC boxes are without *id* fields, and the ratio of vertical box spacing to box height is 1/2.

```
vtoc -i,v50 A.t
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), whatis(1g), yoo(1g), and *gps(5g)*.

NAME

w — who is on and what they are doing

SYNOPSIS

w [**—h**] [**—s**] [**—u**] [*user*]

DESCRIPTION

W prints a summary of the current activity on the system, including what each user is doing. The output looks like:

```
 11:48am  up 14:38,  3 users,  load average: 0.09, 0.37, 1.03
User     tty       login@  idle   JCPU   PCPU  what
root     console   7:11am     9   3:50      5  -sh
chris    tty03     8:36am     2   6:50   1:10  cc main.c
terry    tty05     8:28am         4:37     27  vi calendar
```

The first line gives the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over one, five, and 15 minutes.

The second line gives the field headings: the user's loginname, the name of the **tty** the user is on, the time of day the user logged on, the number of minutes since the user last typed anything, the CPU time used by all processes and their children on that **tty**, the CPU time used by the currently active processes, and the name and arguments of the current process.

OPTIONS

—h The printing of the first two lines (load average and heading) is suppressed.

—s The output is printed in short form. The **tty** is abbreviated; login time, CPU times, and arguments to commands are omitted.

—u Only the first line (load average) is printed.

user
The output is restricted to the specified *user*.

FILES

/etc/utmp	Accounting file.
/dev/cvt	Table of kernel symbols.
/dev/kmem	Image of kernel memory.
/dev/mem	Image of physical memory.
/dev/drum	Image of swap space.

DIAGNOSTICS

Can't read kernel symbols
W could not read kernel symbols from */dev/cvt* (see *cvt(4)*).

No kmem
W could not open */dev/kmem* for reading.

No drum
W could not open */dev/drum* for reading.

No mem
W could not open */dev/mem* for reading.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

The notion of the "current process" is muddy. The current algorithm is "the highest numbered process on the terminal that is not ignoring interrupts; or, if there is none, the highest numbered process on the terminal". This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. (In cases where no process can be found, **w** prints a dash (—).)

The CPU time is only an estimate; in particular, if someone leaves a background process running after logging out, the person currently on that terminal is "charged" with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

W does not know about the new conventions for detection of background jobs. It will sometimes find a background job instead of the right one.

SEE ALSO

who(1n), finger(1n), ps(1), uptime(1n), cvt(4).

NAME

wait — wait for background processes (**csh** built–in)

SYNOPSIS

wait

DESCRIPTION

The **wait** command waits for all running background jobs to either complete or to stop (if *tostop* mode is set on the terminal). If the shell is interactive, an interrupt will disrupt the wait, at which time the job information for all outstanding jobs is printed.

EXAMPLES

The following command waits for all running background processes to complete.

```
wait
```

RETURN VALUE

[0]

CAVEATS

Unlike *wait(1sh)*, **wait** does not take any arguments.

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh), exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1sh), which(1csh), execve(2), wait(2), wait3(2).

NAME

wait — await completion of process (**sh** built–in)

SYNOPSIS

wait [*n*]

DESCRIPTION

Wait waits for the specified process to complete and sets the exit status to that of the waited–for process.

If *n* is not given, all currently active child processes are waited for and the exit status is zero.

Because the *wait(2)* system call must be executed in the parent process, the Shell itself executes **wait**, without creating a new process.

EXAMPLES

The major use of **wait** is in the area of multiprogramming at the shell level. The following shell script takes each file argument and checks it for spelling errors and looks for the word *UTek* (ignoring the case of the letters). All files are processed simultaneously. After all files have been processed, a report is generated for those files with spelling errors and/or that contain the word *UTek*.

```
#!/bin/sh
trap `rm -f /tmp/speller.*.$$ /tmp/spellgrep.*.$$;exit` 0 1 2
for i in "$@"
{
        if test ! -r "$i"
        then
                echo "$0 : $i : Permission denied."
                exit 1
        fi
        spell "$i" > /tmp/speller."$i".$$&
        grep -i utek "$i" > /tmp/spellgrep."$i".$$&
}
wait
for i in "$@"
{
        if test -s /tmp/speller."$i".$$
        then
                echo "Spelling errors were found in the file $i."
        fi
        set `wc -l /tmp/spellgrep."$i".$$`
        if test $1 -ne 0
        then
                echo "The word `utek` was found in the file $i \c"
                echo "on $1 lines."
        fi
}

exit 0
```

It is important to note that background jobs (those started with **&**) do not have their processes printed by the shell unless the shell is interactive. This means that the shell programmer does not have to worry about getting rid of these messages. This process ID is instead available in the variable $!.

RETURN VALUE

The exit status returned by **wait** is the maximum of the exit statuses returned by all of the waited–for processes.

CAVEATS

Not all the processes of a three– or more–stage pipeline are children of the Shell, and thus cannot be waited for. (This bug does not apply to *csh(1csh)*.)

SEE ALSO

break(1sh), cd(1sh), chdir(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), type(1sh), ulimit(1sh), umask(1sh), unset(1sh), which(1sh), execve(2), wait(2).

NAME

wall — write to all users

SYNOPSIS

wall [*filename*]

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by *Broadcast Message* . . . , to all logged in users.

The sender should be superuser to override any protections the users may have invoked.

EXAMPLES

The following example will attempt to broadcast the message contained in the file *mesg* to all logged in users:

```
wall mesg
```

FILES

/dev/tty?	User terminal ports
/etc/utmp	Active login information

DIAGNOSTICS

Cannot send to ...	This message is given when the *open* on a user's *tty* file fails.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.

CAVEATS

The maximum message length is 3000 characters.

SEE ALSO

mesg(1), write(1).

NAME

wc — word count

SYNOPSIS

wc —[**lwc**] [—**v**] [*name...*]

DESCRIPTION

Wc counts lines, words, and characters in the named files, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs or newlines.

If an argument beginning with one of the options —**l**,—**w**, or—**c**, the specified counts (lines, words, or characters) are selected by the letters **l**, **w**, or **c**. The default is —**lwc** unless —**v** is specified.

The —**v** option asks for a verbose output format with headers.

OPTIONS

—**c** Calculates the number of characters in the file.

—**l** Calculates the number of lines in the file.

—**v** Asks for a verbose output format with headers.

—**w**
Calculates the number of words in the file.

EXAMPLES

The following invocation of this command a line of headers labeling each value returned, followed by a line of the actual values for the file *test*:

```
wc -v test
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[P_WARN]	A system error occurred. Execution continues. See *intro(2)* for more information on system errors.

SEE ALSO

ls(1), du(1).

AME

what — identify SCCS files

YNOPSIS

what *filenames*

ESCRIPTION

What searches the given files for all occurrences of the pattern that *get*(1sccs) substitutes for **%Z%** (this is **@(#)** at this printing) and prints out what follows until the first ", >, newline, \, or null character.

XAMPLES

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(1sccs)*, which automatically inserts identifying information, but it can also be used where the information is inserted manually.

)IAGNOSTICS

Use *sccshelp(1sccs)* for explanations.

:AVEATS

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.

;EE ALSO

get(1sccs), sccshelp(1sccs).

NAME

whatis — brief online documentation for UTek Graphics Tools

SYNOPSIS

whatis [**—o**] [*name ...*]

DESCRIPTION

Whatis prints a brief description of each *name* given. If no *name* is given, then the current list of description *names* is printed. The command **whatis** * prints out every description.

OPTIONS

—o just print command options

FILES

*/usr/lib/graf/whatis.d/**	Short descriptive files for the UTek Graphics Tools commands.

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), man(1man), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1man), yoo(1g), and *gps(5g).*

NAME

whatis — whatis database title search

SYNOPSIS

whatis [**—f** *file*] [**—i** *section*] [**—p**] [**—s** *section*] [**—v**] *title ...*

DESCRIPTION

whatis searches whatis database files, as described in *whatis(5man),* for given titles and formats the matching database entries. The titles given must match completely except for case, which is ignored.

If a section is specified with the **—i** option, only entries which do not match the section will be considered. If a section is specified with the **—s** option, only entries which match the section will be considered. The *section* argument consists of a section number and optional subsection text, such as '1' or '1mh'. The section number can be a '+', which means that any section number is allowed, such as '+' (meaning 1–8) or '+mh' (meaning any section number from 1 to 8 followed by the subsection text 'mh'). Also, the subsection text can be replaced by a '+', as in '1+', which matches any section that begins with a 1 (including '1' alone). The case of the subsection text is ignored.

If the **—f** option is not given, the file /usr/lib/man/directories is read to get the locations of the database files that correspond to directories listed in the PATH environment variable, and the file $HOME/.manrc is read to get the locations of personal manual page directories.

The normal format for a line is

page (*section*⟨tabs⟩ – *description*

where *page* is the manual page title, *section* is the manual page section, and *description* is the text from the NAME section of the manual page found after the '–'. The ⟨tabs⟩ field is enough tabs to fill out the first 24 positions on the line.

OPTIONS

—f *file*
Search the named file as a whatis database instead of using the defaults.

—i *section*
Print only entries whose section field does not match the given section specifier.

—p Print only the manual page name and the section for matching entries.

—s *section*
Print only entries whose section field matches the given section specifier.

—v Verbose. Before the matching entries for a given whatis database are printed, print the name of the database with a short description. In the case of databases that correspond to elements of the PATH variable, the path element is noted instead.

EXAMPLES

The following invocation prints the descriptions for all manual pages named *read*.

```
whatis read
```

This invocation prints the names and sections of all manual pages whose name is 'write' and whose section specifier is only a number.

```
whatis -s + write
```

FILES

/usr/lib/man/directories	Manual page search directory information.
$HOME/.manrc	Searched for "personal" manual page directory names.
whatis	The name of the whatis database file.

VARIABLES

PATH	The user's execution path.
HOME	The user's home directory.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.

SEE ALSO

apropos(1man), buildif(1man), help(1man), makewhatis(1man), man(1man), section(1man), man(5man), manindex(5man), whatis(5man), catman(8man).

NAME

which — report command interpretation (**csh** built–in)

SYNOPSIS

which [**—a**] *command* . . .

DESCRIPTION

The command **which** reports the interpretation of the given commands. If the command is an alias, the text "*command* **is an alias for**" followed by the alias expansion, is printed. If the command is a built–in, the text "*command* **is a builtin command**" is printed. Otherwise, the execution path is searched for *command*. If the command is in the path, the text "*command* **is** *pathname*" is printed. If the command can not be found, the text "*command* **is not a command**" is printed.

OPTIONS

—a Print all interpretations of the command instead of just the one that would be executed.

EXAMPLES

Assume that the alias 'echo' exists and is set to 'cat \!*', and that the command /bin/echo exists, and that /bin is in the user's execution path. Also, assume that there is no alias or command in the execution path called 'goober'. In this case, the command

```
which -a echo goober
```

would print

```
echo is an alias for cat \!*
echo is a builtin command
echo is /bin/echo
goober is not a command
```

RETURN VALUE

[NO_ERRS]	Command completed without error.
[1]	An error of the type described in the message occurred.

CAVEATS

This command has a different use than *pathof(1),* and is not intended as a replacement.

The execution path hashing is not taken into account during path searches, so it is possible for **which** to report a pathname which is not the same one used by *csh(1csh).*

SEE ALSO

@(1csh), alias(1csh), bg(1csh), break(1csh), cd(1csh), chdir(1csh), continue(1csh), csh(1csh), dirs(1csh), echo(1csh), eval(1csh), exec(1csh),

exit(1csh), fg(1csh), glob(1csh), goto(1csh), hashstat(1csh), history(1csh), jobs(1csh), kill(1csh), limit(1csh), logout(1csh), nice(1csh), nohup(1csh), notify(1csh), onintr(1csh), pathof(1), popd(1csh), pushd(1csh), rehash(1csh), repeat(1csh), set(1csh), setenv(1csh), sh(1sh), shift(1csh), source(1csh), stop(1csh), suspend(1csh), time(1csh), type(1sh), umask(1csh), unhash(1csh), unalias(1csh), unlimit(1csh), unset(1csh), unsetenv(1csh), wait(1csh), which(1sh).

NAME

type, which — report command interpretation (*sh* built–in)

SYNOPSIS

type [*name*]

DESCRIPTION

Each *name* is described in terms of what would be executed when that command is executed.

Builtin commands are described by a line like : `name is a shell builtin.`

Functions are described by a line like : `name is a function`, followed by the text of the function.

Hashed commands are described by a line like: `name is hashed (pathname)`, where *pathname* is the current known location of the command.

Regular commands are described by a line like : `name is pathname`, where *pathname* is the pathname of the command as found by the execution path.

Commands that do not exist are described by a line like : `name not found.`

The command **which** is a synonym for **type**.

EXAMPLES

To find out what would be executed by typing **ls**, execute the command

```
type ls
```

The command

```
type trap
```

always prints `trap is a shell builtin.`

RETURN VALUE

[NO_ERRS] Command completed without error.

CAVEATS

The commands **type** and *pathof(1)* are similar, but not the same. **Pathof** only knows how to search the execution path and will only print filenames, whereas **type** knows about builtin commands and functions.

SEE ALSO

break(1sh), cd(1sh), continue(1sh), csh(1csh), echo(1sh), eval(1sh), exec(1sh), exit(1sh), export(1sh), hash(1sh), login(1), pathof(1), pwd(1sh), read(1sh), readonly(1sh), return(1sh), set(1sh), sh(1sh), shift(1sh), test(1sh), times(1sh), trap(1sh), ulimit(1sh), umask(1sh), unset(1sh), wait(1sh), execve(2).

NAME

who, users, u — who's logged in on local machines or on the current host system

SYNOPSIS

who [**—q**] [**—s**] [**—H**] [**—T**] [**—N**] [*who-filename*]
who [**am I**]
who [**—r**] [**—H**] [*remote-hosts*]
users

DESCRIPTION

Who, without an argument, lists the loginname, terminal name, login time, and idle time for each current UTek user. The commands **users** and **u** are short forms of who that lists only the names (equivalent to **who -q**).

Without an argument, **who** examines the */etc/utmp* file to obtain its information. If a file is given, that file is examined. Typically the given file will be */usr/adm/wtmp*, which contains a record of all the logins, logouts, and crashes since the creation of the *wtmp* file. Each login is listed with username, terminal name (with */dev/* suppressed), and date and time. When an argument is given, logouts produce a similar line without a username. Reboots produce a line with *x* in the place of the device name and a fossil time indicative of when the system went down.

With the arguments, **am I** (or **am i**) **who** tells who you are logged in as.

The idle time is the time since the last input by the user. If the user is active (idle time ⟨ 1 min) a period will be printed; if the user has left the line hanging (idle time ⟩ 24 hrs) the word "old" will be printed. It can be suppressed with the **—s** switch.

The **—r** switch without any argument polls all machines on the local network, and prints the output. Only machines which provide the **systat/udp** service will answer (on UTek this service is provided by *udpd*). If a list of hosts is given then they are sequentially read using the **systat/tcp** service (on UTek this service is usually provided by *tcpd*).

OPTIONS

—r*[host(s)]*
Report users on remote systems, if none given then use broadcasts. This is the **rwho** command.

—q Report only usernames separated by spaces. This is the *users(1n)* command.

—H
Print out a header before the output.

—s Short form, don't check or print idle time.

—T Print status of tty:

+	writeable
-	not writeable
x	exclusive open
?	hung (no carrier)

—N
Network who (i.e. run by tcpd); print carriage return/line feed after each name (for non–UNIX type hosts).

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.

CAVEATS

The format of the remote who output may vary, no standard format is required.

SEE ALSO

utmp(5), users(1n), udpd(8n), tcpd(8n).

NAME

whoami — print effective current user ID

SYNOPSIS

whoami

DESCRIPTION

Whoami prints the effective user ID which is running the current process. This is different from **who am i**, which prints the name of the person logged in on the **tty**.

EXAMPLES

The following shell script fragment prevents the user ID *root* from executing the command:

```
#!/bin/sh
if test 'root' = "`whoami`"
then
        echo "This shell script may not be run by 'root'."
        exit 1
fi
...
```

FILES

/etc/passwd	Name data base.

RETURN VALUE

[NO_ERRS]	Command completed without error.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.

CAVEATS

The user ID returned corresponds to who you are at that point in time. So if you have used **su** to substitute another username, that name is returned.

SEE ALSO

who(1n), su(1).

NAME

write — write to another user

SYNOPSIS

write *[-v]*user(s)...
write *user ttyname*

DESCRIPTION

Write sends messages from your terminal to another user. The *user* can be either a local user or user@host in which case the message is sent or the network. **Write** operates in two modes. If **write** is called with more than one user or the user is remote, **write** will read the whole message before sending it. With one argument which is a local user or two arguments the second of which is a ttyname, **write** copies lines from your terminal to that of another user.

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 *ttyname* argument may be used to indicate the appropriate terminal name.

In either mode, before printing the message, write will print a header

```
Message from yourname@yourhost on yourttyname...
```

Permission to write may be denied or granted by use of the **mesg** command. At the outset writing is allowed. Certain commands, in particular *nroff(1)* and *pr(1)* disallow messages in order to prevent messy output.

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** in line at a time mode: when you first write to another user, wait for him or her to write back before starting to send. Each party should end each message with a distinctive signal that the other may reply to:

(o) This is conventional for "over".

(oo) This means "over and out", and is suggested when conversation is about to be terminated.

OPTIONS

—v **Write** when in "message" mode will show each user that receives the message.

EXAMPLES

The following example simulates a session with the user **johndoe**:

```
write johndoe
(johndoe initiates a write to you and acknowledges your write)
Message from johndoe on tty?? at 10:15 ...
(You type your message.)
  ...
(o)
(johndoe possibly responds and signals that conversation be te
  ...
(oo)
(You agree to terminate conversation.)
(oo)
(You exit write by sending end-of-file.)
^D
```

The following shows sending a message to a user **john** on another host **fuzzy**:

```
write john@fuzzy
Enter message, terminated with ^D or '.'
(You type your message.)
^D
-Eot-
```

FILES

/etc/utmp	To find user.
/bin/sh	To execute the **!** character.

RETURN VALUE

[USAGE]	Incorrect command line syntax. Execution terminated.
[P_ERR]	A system error occurred. Execution terminated. See *intro(2)* for more information on system errors.
[NP_ERR]	An error occurred that was not a system error. Execution terminated.
[INTERNAL]	An unexpected error occurred. Execution was terminated. Record the message and save the core file for analysis. Contact service personnel at your Tektronix field office.
[NP_WARN]	An error warranting a warning message occurred. Execution continues.

CAVEATS

The remote host must have an SMTP server that allows the SEND command. This function is not part of 4.2BSD UNIX sendmail, but is provided with UTek sendmail.

To prevent "letter-bombs", **write** silently eats control characters other than tab and backspace.

In message mode, messages are limited to 4 kbytes.

SEE ALSO

mail(1mh), mesg(1), who(1n), talk(1n).

NAME

xargs — construct argument list(s) and execute command

SYNOPSIS

xargs [—**e***eofstr*][—**i***replstr*][—**l***number*][—**n***number*][—**p**] [—**s***size*][—**t**][—**x**] [*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 continuous strings of characters delimited by one or more blanks, tabs, or newlines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (exception: see —**i** flag). Flags —**i**, —**l**, and —**n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated arguments. This process is repeated until there are no more arguments. When there are flag conflicts (for example, —**l** vs. —**n**), the last flag has precedence.

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(1sh)*) with an appropriate value to avoid accidentally returning with —**1**.

OPTIONS

—**e***eofstr*

Eofstr is taken as the logical end–of–file string. Underscore (_) is assumed for the logical **EOF** string if —**e** is not coded. The —**e** option with no *eofstr* coded turns off the logical **EOF** string capability (underscore is taken literally). **Xargs** reads standard input until either end–of–file or the logical **EOF** string is encountered.

—**i***replstr*

Insert mode: *command* is executed for each line from standard input, taking the entire line as a single argument, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of five 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.

—l*number*
Command is executed for each nonempty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first newline *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next nonempty line. If *number* is omitted 1 is assumed. Option **—x** is forced.

—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.

—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**, meaning yes, (optionally followed by anything), will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.

—s*size*
The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 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.

—t Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

—x Causes **xargs** to terminate if any argument list would be greater than *size* characters; **—x** is forced by the options **—i** and **—l**. When none of the options **—i**, **—l**, or **—n** are coded, the total length of all arguments must be within the *size* limit.

EXAMPLES

The following will move all files from directory *$1* to directory *$2*, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{ } $2/{ }
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end–of–file *log*:

```
(logname; date; echo $0 $*) ¦ xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* either one at a time, or many at a time:

```
1.   ls ¦ xargs -p -1 ar r arch
2.   ls ¦ xargs -p -1 ¦ xargs ar r arch
```

The following will execute *diff(1)* with successive pairs of arguments originally typed as shell arguments:

```
echo $* ¦ xargs -n2 diff
```

SEE ALSO

sh(1sh), echo(1), find(1).

NAME

xstr — extract strings from C programs to implement shared strings

SYNOPSIS

xstr [**—c**] [—] [*filename*]

DESCRIPTION

Xstr maintains a file *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read–only. The command

xstr —c *filename*

will extract the strings from the C source in *filename*, replacing string references by expressions of the form **&xstr**[*number*]) for some number. An appropriate declaration of **xstr** is prepended to the file. The resulting C text is placed in the file *x.c*, to then be compiled. The strings from this file are placed in the *strings* database if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the database.

After all components of a large program have been compiled, a file *xs.c* declaring the common **xstr** space can be created by a command of the form

xstr

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read–only (shared) saving space and swap overhead.

Xstr can also be used on a single file. A command

xstr *filename*

creates the files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run **xstr** after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. **Xstr** reads from its standard input when the dash argument (—) is given. An appropriate command sequence for running **xstr** after the C preprocessor is:

```
cc -E name.c ¦ xstr -c -
cc -c x.c
mv x.o name.o
```

Xstr does not touch the file *strings* unless new items are added; thus, **make** can avoid remaking *xs.o* unless truly necessary.

OPTIONS

—c **Xstr** extracts strings from the C source in the given file.

— **Xstr** reads from the standard input.

FILES

strings	Database of strings.
x.c	Massaged C source.
xs.c	C source for definition of array **xstr**.
*/tmp/xs**	Temporary file when **xstr** *filename* doesn't touch *strings*.

CAVEATS

If a string is a suffix of another string in the database, but the shorter string is seen first by **xstr**, both strings will be placed in the database, when just placing the longer one there will do.

SEE ALSO

cc(1), mv(1).

NAME

yacc — yet another compiler–compiler

SYNOPSIS

yacc [**—vd**] *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**.

OPTIONS

—d 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.

—v 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.

FILES

y.output	Contains description of parsing tables and conflict reports.
y.tab.c	Output.
y.tab.h	Defines for token names.
yacc.tmp, yacc.acts	Temporary files.
/usr/lib/yaccpar	Parser prototype for C programs.

DIAGNOSTICS

The number of *reduce-reduce* and *shift-reduce* conflicts is reported on the standard 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.

CAVEATS

Because filenames are fixed, at most, one **yacc** process can be active in a given directory at a time.

SEE ALSO

lex(1).

NAME

yes — be repetitively affirmative

SYNOPSIS

yes [*expletive*]

DESCRIPTION

Yes repeatedly outputs *y*, or if *expletive* is given, that is output repeatedly. Termination is by interrupt.

EXAMPLES

The following example repeatedly prints "This is a test.".

```
yes "This is a test."
```

RETURN VALUE

[?] This utility never terminates normally. Therefore, its return value is nondeterministic.

NAME

yoo — pipe fitting

SYNOPSIS

yoo *file*

DESCRIPTION

Yoo is a piping primitive that deposits the output of a pipeline into a *file* used in the pipeline. Note that, without **yoo**, this is not usually successful as it causes a read and write on the same file simultaneously.

EXAMPLES

The following example stores the concatenation of files A and B in B.

```
cat A B ¦ yoo B
```

SEE ALSO

abs(1g), af(1g), bar(1g), bel(1g), bucket(1g), ceil(1g), cor(1g), cusum(1g), cvrtopt(1g), dtoc(1g), erase(1g), exp(1g), floor(1g), gamma(1g), gas(1g), gd(1g), ged(1g), graphics(1g), gtop(1g), hardcopy(1g), hilo(1g), hist(1g), hpd(1g), intro(1g), label(1g), list(1g), log(1g), lreg(1g), mean(1g), mod(1g), pair(1g), pd(1g), pie(1g), plot(1g), point(1g), power(1g), prime(1g), prod(1g), ptog(1g), qsort(1g), quit(1g), rand(1g), rank(1g), remcom(1g), root(1g), round(1g), siline(1g), sin(1g), subset(1g), td(1g), tekset(1g), title(1g), total(1g), ttoc(1g), var(1g), vtoc(1g), whatis(1g), and *gps(5g).*