awk

awk - pattern scanning and processing language

SYNOPSIS

```
/usr/bin/awk [ -f progfile ] [ -Fc ] [ 'prog' ]
     [ parameters ] [ filename...]

/usr/xpg4/bin/awk [ -F ERE ] [ -v assignment ... ]
     'program' | -f progfile ...  [ argument ... ]
```

DESCRIPTION

The /usr/xpg4/bin/awk utility is described on the nawk(1) manual page.

The /usr/bin/awk utility scans each input filename for lines that match any of a set of patterns specified in prog. The prog string must be enclosed in single quotes (') to protect it from the shell. For each pattern in prog there may be an associated action performed when a line of a filename matches the pattern. The set of pattern-action statements may appear literally as prog or in a file specified with the -f progfile option. Input files are read in order; if there are no files, the standard input is read. The file name '-' means the standard input.

OPTIONS

-f progfile	awk uses the set of patterns it reads from progfile.
-Fc	Use the character c as the field separator (FS) character. See the discussion of FS below.

USAGE

Input Lines

Each input line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern. Any filename of the form var=value is treated as an assignment, not a filename, and is executed at the time it would have been opened if it were a filename. Variables assigned in this manner are not available inside a BEGIN rule, and are assigned after previ-

ously specified files have been read.

An input line is normally made up of fields separated by white spaces. (This default can be changed by using the FS built-in variable or the -Fc option.) The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value that does not include any of the white spaces, then leading blanks are not ignored. The fields are denoted $1, $2, ...; $0 refers to the entire line.

Pattern-action Statements

A pattern-action statement has the form:

```
pattern { action }
```

Either pattern or action may be omitted. If there is no action, the matching line is printed. If there is no pattern, the action is performed on every input line. Pattern-action statements are separated by newlines or semicolons.

Patterns are arbitrary Boolean combinations (!, ||, &&, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

```
expression relop expression
expression matchop regular_expression
```

where a relop is any of the six relational operators in C, and a matchop is either ~ (contains) or !~ (does not contain). An expression is an arithmetic expression, a relational expression, the special expression

```
var in array
```

or a Boolean combination of these.

Regular expressions are as in egrep(1). In patterns they must be surrounded by slashes. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between the occurrence of the first pattern to the occurrence of the second pattern.

The special patterns BEGIN and END may be used to capture

control before the first input line has been read and after the last input line has been read respectively. These keywords do not combine with any other patterns.

Built-in Variables

Built-in variables include:

FILENAME	name of the current input file
FS	input field separator regular expression (default blank and tab)
NF	number of fields in the current record
NR	ordinal number of the current record
OFMT	output format for numbers (default %.6g)
OFS	output field separator (default blank)
ORS	output record separator (default newline)
RS	input record separator (default newline)

An action is a sequence of statements. A statement may be one of the following:

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

Statements are terminated by semicolons, newlines, or right braces. An empty expression-list stands for the whole input line. Expressions take on string or numeric values as appropriate, and are built using the operators +, -, *, /,

%, ^ and concatenation (indicated by a blank). The operators ++, --, +=, -=, *=, /=, %=, ^=, >, >=, <, <=, ==, !=, and ?: are also available in expressions. Variables may be scalars, array elements (denoted x[i]), or fields. Variables are initialized to the null string or zero. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (""), with the usual C escapes recognized within.

The print statement prints its arguments on the standard output, or on a file if >expression is present, or on a pipe if '|cmd' is present. The output resulted from the print statement is terminated by the output record separator with each argument separated by the current output field separator. The printf statement formats its expression list according to the format (see printf(3S)).

Built-in Functions

The arithmetic functions are as follows:

cos(x)	Return cosine of x, where x is in radians.
sin(x)	Return sine of x, where x is in radians.
exp(x)	Return the exponential function of x.
log(x)	Return the natural logarithm of x.
sqrt(x)	Return the square root of x.
int(x)	Truncate its argument to an integer. It will be truncated toward 0 when x > 0.

The string functions are as follows:

index(s, t)	Return the position in string s where string t first occurs, or 0 if it does not occur at all.
int(s)	truncates s to an integer value. If s is not specified, $0 is used.
length(s)	Return the length of its argument taken as a string, or of the whole line if there is no argument.

match(s, re) Return the position in string s where the regular expression re occurs, or 0 if it does not occur at all.

split(s, a, fs)
Split the string s into array elements a[1], a[2], a[n], and returns n. The separation is done with the regular expression fs or with the field separator FS if fs is not given.

sprintf(fmt, expr, expr,...)
Format the expressions according to the printf(3S) format given by fmt and returns the resulting string.

substr(s, m, n)
returns the n-character substring of s that begins at position m.

The input/output function is as follows:

getline Set $0 to the next input record from the current input file. getline returns 1 for successful input, 0 for end of file, and -1 for an error.

Large File Behavior

See largefile(5) for the description of the behavior of awk when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

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

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN    { FS = ",[ \t]*|[ \t]+" }
     { print $2, $1 }
```

Add up first column, print sum and average:

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

Print fields in reverse order:

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

Print all lines between start/stop pairs:

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

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

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

Print a file, filling in page numbers starting at 5:

```
/Page/    { $2 = n++; }
     { print }
```

Assuming this program is in a file named prog, the following command line prints the file input numbering its pages starting at 5: awk -f prog n=5 input.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of awk: LC_CTYPE and LC_MESSAGES.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/awk

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWesu
CSI	Enabled

/usr/xpg4/bin/awk

ATTRIBUTE TYPE	ATTRIBUTE VALUE

Availability	SUNWxcu4
CSI	Enabled

SEE ALSO

egrep(1), grep(1), nawk(1), sed(1), printf(3S), attributes(5), environ(5), largefile(5), xpg4(5)

NOTES

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

bc

bc - arbitrary precision arithmetic language

SYNOPSIS

bc [-c] [-l] [file...]

DESCRIPTION

The bc utility implements an arbitrary precision calculator. It takes input from any files given, then reads from the standard input. If the standard input and standard output to bc are attached to a terminal, the invocation of bc is interactive , causing behavioural constraints described in the following sections. bc processes a language that resembles C and is a preprocessor for the desk calculator program dc, which it invokes automatically unless the -c option is specified. In this case the dc input is sent to the standard output instead.

USAGE

The syntax for bc programs is as follows:

L means a letter a-z,

E means an expression: a (mathematical or logical) value, an operand that takes a value, or a combination of operands and operators that evaluates to a value,

S means a statement.

Comments

Enclosed in /* and */.

Names (Operands)

Simple variables: L.
Array elements: L [E] (up to BC_DIM_MAX dimensions).
The words ibase, obase (limited to BC_BASE_MAX), and scale (limited to BC_SCALE_MAX).

Other Operands

Arbitrarily long numbers with optional sign and decimal point.

```
        Strings of fewer than BC_STRING_MAX characters,
        between double quotes (").
        ( E )

        sqrt ( E )                  Square root

        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
   .string

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

Functions in -l Math Library
        s(x)    sine
        c(x)    cosine
        e(x)    exponential
```

```
l(x)    log
a(x)    arctangent
j(n,x)  Bessel function
```

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or new-lines may separate statements. Assignment to scale influences the number of digits to be retained on arithmetic operations in the manner of dc. Assignments to ibase or obase set the input and output number radix respectively.
The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. auto variables are stacked during function calls. When using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

OPTIONS

-c Compile only. The output is dc commands that are sent to the standard output.

-l Define the math functions and initialize scale to 20, instead of the default zero.

OPERANDS

The following operands are supported:

file A pathname of a text file containing bc program statements. After all cases of file have been read, bc will read the standard input.

EXAMPLES

In the shell, the following assigns an approximation of the first ten digits of - n to the variable x :

```
x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

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
            }
       }
```

Prints approximate values of the exponential function of the first ten integers:

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

or

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

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of bc: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 All input files were processed successfully.

unspecified An error occurred.

FILES

/usr/lib/lib.b	mathematical library
/usr/include/limits.h	to define BC_ parameters

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWesu

SEE ALSO

dc(1), awk(1), attributes(5)

NOTES

The bc command does not recognize the logical operators && and ||.

The for statement must have all three expressions (E's).

chgrp

chgrp - change file group ownership

SYNOPSIS

chgrp [-fhR] group file

DESCRIPTION

The chgrp utility will set the group ID of the file named by each file operand to the group ID specified by the group operand.

For each file operand, it will perform actions equivalent to the chown(2) function, called with the following arguments:

- The file operand will be used as the path argument.
- The user ID of the file will be used as the owner argument.
- The specified group ID will be used as the group argument.

Unless chgrp is invoked by a process with appropriate privileges, the set-user-ID and set-group-ID bits of a regular file will be cleared upon successful completion; the set-user-ID and set-group-ID bits of other file types may be cleared.

The operating system has a configuration option {_POSIX_CHOWN_RESTRICTED}, to restrict ownership changes. When this option is in effect, the owner of the file may change the group of the file only to a group to which the owner belongs. Only the super-user can arbitrarily change owner IDs, whether or not this option is in effect. To set this configuration option, include the following line in /etc/system:

```
set rstchown = 1
```

To disable this option, include the following line in /etc/system:

```
set rstchown = 0
```

{_POSIX_CHOWN_RESTRICTED} is enabled by default. See system(4) and fpathconf(2).

OPTIONS

-f Force. Do not report errors.

-h If the file is a symbolic link, change the group of the symbolic link. Without this option, the group of the file referenced by the symbolic link is changed.

-R Recursive. chgrp descends through the directory, and any subdirectories, setting the specified group ID as it proceeds. When a symbolic link is encountered, the group of the target file is changed (unless the -h option is specified), but no recursion takes place.

OPERANDS

The following operands are supported:

group A group name from the group database or a numeric group ID. Either specifies a group ID to be given to each file named by one of the file operands. If a numeric group operand exists in the group database as a group name, the group ID number associated with that group name is used as the group ID.

file A path name of a file whose group ID is to be modified.

USAGE

See largefile(5) for the description of the behavior of chgrp when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of chgrp: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

FILES

/etc/group group file

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled (see NOTES)

SEE ALSO

chmod(1), chown(1), id(1M), chown(2), fpathconf(2), group(4), passwd(4), system(4), attributes(5), environ(5), largefile(5)

NOTES

chgrp is CSI-enabled except for the group name.

chmod

chmod - change the permissions mode of a file

SYNOPSIS

```
chmod [ -fR ] <absolute-mode> file...
chmod [ -fR ] <symbolic-mode-list> file...
```

DESCRIPTION

chmod changes or assigns the mode of a file. The mode of a file specifies its permissions and other attributes. The mode may be absolute or symbolic.

Absolute mode

An absolute mode is specified using octal numbers:

```
chmod nnnn file ...
```

where:

n — a number from 0 to 7. An absolute mode is constructed from the OR of any of the following modes:

4000	Set user ID on execution.
20#0	Set group ID on execution if # is 7, 5, 3, or 1. Enable mandatory locking if # is 6, 4, 2, or 0. For directories, files are created with BSD semantics for propagation of the group ID. With this option, files and subdirectories created in the directory inherit the group ID of the directory, rather than of the current process. It may be cleared only by using symbolic mode.
1000	Turn on sticky bit. See chmod(2).
0400	Allow read by owner.
0200	Allow write by owner.
0100	Allow execute (search in directory) by owner.
0700	Allow read, write, and execute

(search) by owner.

0040 Allow read by group.

0020 Allow write by group.

0010 Allow execute (search in directory) by group.

0070 Allow read, write, and execute (search) by group.

0004 Allow read by others.

0002 Allow write by others.

0001 Allow execute (search in directory) by others.

0007 Allow read, write, and execute (search) by others.

Note that the setgid bit cannot be set (or cleared) in absolute mode; it must be set (or cleared) in symbolic mode using g+s (or g-s).

Symbolic mode

A symbolic mode specification has the following format:

chmod <symbolic-mode-list> file...

where: <symbolic-mode-list> is a comma-separated list (with no intervening whitespace) of symbolic mode expressions of the form:

[who] operator [permissions]

Operations are performed in the order given. Multiple permissions letters following a single operator cause the corresponding operations to be performed simultaneously.

who zero or more of the characters u, g, o, and a specifying whose permissions are to be changed or assigned:

u user's permissions

g group's permissions

o others' permissions

a all permissions (user, group, and other)

If who is omitted, it defaults to a, but the setting of the file mode creation mask (see umask in sh(1) or csh(1) for more information) is taken into account. When who is omitted, chmod will not override the restric-

tions of your user mask.

operator either +, -, or =, signifying how permissions are to be changed:

+ Add permissions.

If permissions is omitted, nothing is added.

If who is omitted, add the file mode bits represented by permissions, except for the those with corresponding bits in the file mode creation mask.

If who is present, add the file mode bits represented by the permissions.

- Take away permissions.

If permissions is omitted, do nothing.

If who is omitted, clear the file mode bits represented by permissions, except for those with corresponding bits in the file mode creation mask.

If who is present, clear the file mode bits represented by permissions.

= Assign permissions absolutely.

If who is omitted, clear all file mode bits; if who is present, clear the file mode bits represented by who.

If permissions is omitted, do nothing else.

If who is omitted, add the file mode bits represented by permis-

sions, except for the those with corresponding bits in the file mode creation mask.

If who is present, add the file mode bits represented by permissions.

Unlike other symbolic operations, = has an absolute effect in that it resets all other bits represented by who. Omitting permissions is useful only with = to take away all permissions.

permission

any compatible combination of the following letters:

r	read permission
w	write permission
x	execute permission
l	mandatory locking
s	user or group set-ID
t	sticky bit
u,g,o	indicate that permission is to be taken from the current user, group or other mode respectively.

Permissions to a file may vary depending on your user identification number (UID) or group identification number (GID). Permissions are described in three sequences each having three characters:

User	Group	Other
rwx	rwx	rwx

This example (user, group, and others all have permission to read, write, and execute a given file) demonstrates two categories for granting permissions: the access class and the permissions themselves.

The letter s is only meaningful with u or g, and t only works with u.

Mandatory file and record locking (l) refers

to a file's ability to have its reading or writing permissions locked while a program is accessing that file.

In a directory which has the set-group-ID bit set (reflected as either -----s--- or -----l--- in the output of 'ls -ld'), files and subdirectories are created with the group-ID of the parent directory-not that of current process.

It is not possible to permit group execution and enable a file to be locked on execution at the same time. In addition, it is not possible to turn on the set-group-ID bit and enable a file to be locked on execution at the same time. The following examples, therefore, are invalid and elicit error messages:

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

Only the owner of a file or directory (or the super-user) may change that file's or directory's mode. Only the super-user may set the sticky bit on a non-directory file. If you are not super-user, chmod will mask the sticky-bit but will not return an error. In order to turn on a file's set-group-ID bit, your own group ID must correspond to the file's and group execution must be set.

OPTIONS

The following options are supported:

-f Force. chmod will not complain if it fails to change the mode of a file.

-R Recursively descend through directory arguments, setting the mode for each file as described above. When symbolic links are encountered, the mode of the target file is changed, but no recursion takes place.

OPERANDS

The following operands are supported:

mode — Represents the change to be made to the file mode bits of each file named by one of the file operands; see DESCRIPTION.

file — A path name of a file whose file mode bits are to be modified.

USAGE

See largefile(5) for the description of the behavior of chmod when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

Deny execute permission to everyone:

```
example% chmod a-x file
```

Allow only read permission to everyone:

```
example% chmod 444 file
```

Make a file readable and writable by the group and others:

```
example% chmod go+rw file
example% chmod 066 file
```

Cause a file to be locked during access:

```
example% chmod +l file
```

Allow everyone to read, write, and execute the file and turn on the set group-ID.

```
example% chmod a=rwx,g+s file
example% chmod 2777 file
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of chmod: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 Successful completion.
>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	enabled

SEE ALSO

ls(1), chmod(2), attributes(5), environ(5), largefile(5)

NOTES

Absolute changes don't work for the set-group-ID bit of a directory. You must use g+s or g-s.

chmod permits you to produce useless modes so long as they are not illegal (for instance, making a text file executable). chmod does not check the file type to see if mandatory locking is meaningful.

If the filesystem is mounted with the nosuid option, setuid execution is not allowed.

chown

chown - change file ownership

SYNOPSIS

chown [-fhR] owner[:group] file...

DESCRIPTION

The chown utility will set the user ID of the file named by each file to the user ID specified by owner, and, optionally, will set the group ID to that specified by group.

If chown is invoked by other than the super-user, the set-user-ID bit is cleared.

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

The operating system has a configuration option {_POSIX_CHOWN_RESTRICTED}, to restrict ownership changes. When this option is in effect the owner of the file is prevented from changing the owner ID of the file. Only the super-user can arbitrarily change owner IDs whether or not this option is in effect. To set this configuration option, include the following line in /etc/system:

```
set rstchown = 1
```

To disable this option, include the following line in /etc/system:

```
set rstchown = 0
```

{_POSIX_CHOWN_RESTRICTED} is enabled by default. See system(4) and fpathconf(2).

OPTIONS

The following options are supported:

-f Do not report errors.

-h If the file is a symbolic link, change the owner of the symbolic link. Without this option, the owner of the

file referenced by the symbolic link is changed.

-R Recursive. chown descends through the directory, and any subdirectories, setting the ownership ID as it proceeds. When a symbolic link is encountered, the owner of the target file is changed (unless the -h option is specified), but no recursion takes place.

OPERANDS

The following operands are supported:

owner[:group] A user ID and optional group ID to be assigned to file. The owner portion of this operand must be a user name from the user database or a numeric user ID. Either specifies a user ID to be given to each file named by file. If a numeric owner exists in the user database as a user name, the user ID number associated with that user name will be used as the user ID. Similarly, if the group portion of this operand is present, it must be a group name from the group database or a numeric group ID. Either specifies a group ID to be given to each file. If a numeric group operand exists in the group database as a group name, the group ID number associated with that group name will be used as the group ID.

file A path name of a file whose user ID is to be modified.

USAGE

See largefile(5) for the description of the behavior of chown when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of chown: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 The utility executed successfully and all requested

changes were made.

>0 An error occurred.

FILES

/etc/passwd system password file

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled (see NOTES)

SEE ALSO

chgrp(1), chmod(1), chown(2), fpathconf(2), passwd(4), system(4), attributes(5), environ(5), largefile(5)

NOTES

chown is CSI-enabled except for the owner and group names.

cp

cp - copy files

SYNOPSIS

```
/usr/bin/cp [-fip] source_file target_file
/usr/bin/cp [-fip] source_file... target
/usr/bin/cp -r|-R [-fip] source_dir... target

/usr/xpg4/bin/cp [-fip] source_file target_file
/usr/xpg4/bin/cp [-fip] source_file... target
/usr/xpg4/bin/cp -r|-R [-fip] source_dir... target
```

DESCRIPTION

In the first synopsis form, neither source_file nor target_file are directory files, nor can they have the same name. The cp utility will copy the contents of source_file to the destination path named by target_file. If target_file exists, cp will overwrite its contents, but the mode (and ACL if applicable), owner, and group associated with it are not changed. The last modification time of target_file and the last access time of source_file are set to the time the copy was made. If target_file does not exist, cp creates a new file named target_file that has the same mode as source_file except that the sticky bit is not set unless the user is superuser; the owner and group of target_file are those of the owner. If target_file is a link to another file with links, the other links remain and target_file becomes a new file.

In the second synopsis form, one or more source_files are copied to the directory specified by target. For each source_file specified, a new file with the same mode (and ACL if applicable), is created in target; the owner and group are those of the user making the copy. It is an error if any source_file is a file of type directory, if target either does not exist or is not a directory.

In the third synopsis form, one or more directories specified by source_dir are copied to the directory specified by target. Either -r or -R must be specified. For each source_dir, cp will copy all files and subdirectories.

OPTIONS

The following options are supported for both /usr/bin/cp and /usr/xpg4/bin/cp:

-f Unlink. If a file descriptor for a destination file cannot be obtained, attempt to unlink the destination file and proceed.

-i Interactive. cp will prompt for confirmation whenever the copy would overwrite an existing target. A y answer means that the copy should proceed. Any other answer prevents cp from overwriting target.

-r Recursive. cp will copy the directory and all its files, including any subdirectories and their files to target.

-R Same as -r, except pipes are replicated, not read from.

/usr/bin/cp

The following option is supported for /usr/bin/cp only:

-p Preserve. cp duplicates not only the contents of source_file, but also preserves the owner and group id, permissions modes, modification and access time, and ACLs if applicable. Note that the command may fail if ACLs are copied to a file system that does not support ACLs. The command will not fail if unable to preserve modification and access time or permission modes. If unable to preserve owner and group id, cp will not fail, and it will clear S_ISUID and S_ISGID bits in the target. cp will print a diagnostic message to stderr and return a non-zero exit status if unable to clear these bits.

In order to preserve the owner and group id, permission modes, and modification and access times, users must have the appropriate file access permissions; this includes being superuser or the same owner id as the destination file.

/usr/xpg4/bin/cp

The following option is supported for /usr/xpg4/bin/cp only:

-p Preserve. cp duplicates not only the contents of source_file, but also preserves the owner and group id,

permission modes, modification and access time, and ACLs if applicable. Note that the command may fail if ACLs are copied to a file system that does not support ACLs. If unable to duplicate the modification and access time or the permission modes, cp will print a diagnostic message to stderr and return a non-zero exit status. If unable to preserve owner and group id, cp will not fail, and it will clear S_ISUID and S_ISGID bits in the target. cp will print a diagnostic message to stderr and return a non-zero exit status if unable to clear these bits.

In order to preserve the owner and group id, permission modes, and modification and access times, users must have the appropriate file access permissions; this includes being superuser or the same owner id as the destination file.

OPERANDS

The following operands are supported:

source_file — A path name of a regular file to be copied.

source_dir — A path name of a directory to be copied.

target_file — A pathname of an existing or non-existing file, used for the output when a single file is copied.

target — A pathname of a directory to contain the copied files.

USAGE

See largefile(5) for the description of the behavior of cp when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

1. To copy a file:

```
example% cp goodies goodies.old
example% ls goodies*
goodies goodies.old
```

2. To copy a list of files to a destination directory:

```
example% cp ~/src/*   /tmp
```

3. To copy a directory, first to a new, and then to an existing destination directory:

```
example% ls ~/bkup
/usr/example/fred/bkup not found
example% cp -r ~/src ~/bkup
example% ls -R ~/bkup
x.c y.c z.sh
example% cp -r ~/src ~/bkup
example% ls -R ~/bkup
src x.c y.c z.sh

src:
x.c y.c z.sh
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of cp: LC_COLLATE, LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 All files were copied successfully.

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/cp

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	enabled

/usr/xpg4/bin/cp

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	enabled

|_____|_____|

SEE ALSO

chmod(1), chown(1), setfacl(1), utime(2), attributes(5), environ(5), largefile(5), xpg4(5)

NOTES

The permission modes of the source file are preserved in the copy.

A -- permits the user to mark the end of any command line options explicitly, thus allowing cp to recognize filename arguments that begin with a -.
If a -- and a - both appear on the same command line, the second will be interpreted as a filename.

cron

cron - clock daemon

SYNOPSIS

/usr/sbin/cron

DESCRIPTION

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

cron only examines crontab or at command files during its own process initialization phase and when the crontab or at command is run. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

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

cron captures the output of the job's stdout and stderr streams, and, if it is non-empty, mails the output to the user. If the job does not produce output, no mail is sent to the user (unless the job is an at(1) job and the -m option was specified when the job was submitted).

Setting cron Defaults

To keep a log of all actions taken by cron, CRONLOG=YES (by default) must be specified in the /etc/default/cron file. If CRONLOG=NO is specified, no logging is done. Keeping the log is a user configurable option since cron usually creates huge log files.

The PATH for user cron jobs can be set using PATH= in /etc/default/cron. The PATH for root cron jobs can be set using SUPATH= in /etc/default/cron. The security implications of setting PATH and SUPATH should be carefully con-

sidered.

Example /etc/default/cron file:

```
CRONLOG=YES
PATH=/usr/bin:/usr/ucb:
```

This example enables logging and sets the default PATH used by non-root jobs to /usr/bin:/usr/ucb:. Root jobs will continue to use /usr/sbin:/usr/bin.

/etc/cron.d/logchecker is a script that checks to see if the log file has exceeded the system ulimit. If so, the log file is moved to /var/cron/olog.

FILES

/etc/cron.d	main cron directory
/etc/cron.d/FIFO	used as a lock file
/etc/default/cron	contains cron default settings
/var/cron/log	cron history information
/var/spool/cron	spool area
/etc/cron.d/logchecker	moves log file to /var/cron/olog if log file exceeds system ulimit.
/etc/cron.d/queuedefs	queue description file for at, batch, and cron.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

SEE ALSO

at(1), crontab(1), sh(1), queuedefs(4), attributes(5)

DIAGNOSTICS

A history of all actions taken by cron is stored in /var/cron/log and (possibly) /var/cron/olog.

csh

csh - shell command interpreter with a C-like syntax

SYNOPSIS

csh [-bcefinstvVxX] [argument...]

DESCRIPTION

csh, the C shell, is a command interpreter with a syntax reminiscent of the C language. It provides a number of convenient features for interactive use that are not available with the Bourne shell, including filename completion, command aliasing, history substitution, job control, and a number of built-in commands. As with the Bourne shell, the C shell provides variable, command and filename substitution.

Initialization and Termination

When first started, the C shell normally performs commands from the .cshrc file in your home directory, provided that it is readable and you either own it or your real group ID matches its group ID. If the shell is invoked with a name that starts with `-', as when started by login(1), the shell runs as a login shell.

If the shell is a login shell, this is the sequence of invocations: First, commands in /etc/.login are executed. Next, commands from the .cshrc file your home directory are executed. Then the shell executes commands from the .login file in your home directory; the same permission checks as those for .cshrc are applied to this file. Typically, the .login file contains commands to specify the terminal type and environment. (For an explanation of file interpreters, see below "Command Execution" and exec(2).)

As a login shell terminates, it performs commands from the .logout file in your home directory; the same permission checks as those for .cshrc are applied to this file.

Interactive Operation

After startup processing is complete, an interactive C shell begins reading commands from the terminal, prompting with hostname% (or hostname# for the privileged user). 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 history list and then parsed, as described under USAGE, below. Finally, the shell executes each command in the current line.

Noninteractive Operation

When running noninteractively, the shell does not prompt for input from the terminal. A noninteractive C shell can execute a command supplied as an argument on its command line, or interpret commands from a file, also known as a script.

OPTIONS

-b Force a "break" from option processing. Subsequent command line arguments are not interpreted as C

to a script without confusion. The shell does not run set-user-ID or set-group-ID scripts unless this option is present.

-c Execute the first argument (which must be present). Remaining arguments are placed in argv, the argument-list variable, and passed directly to csh.

-e Exit if a command terminates abnormally or yields a nonzero exit status.

-f Fast start. Read neither the .cshrc file, nor the .login file (if a login shell) upon startup.

-i Forced interactive. Prompt for command line input, even if the standard input does not appear to be a terminal (character-special device).

-n Parse (interpret), but do not execute commands. This option can be used to check C shell scripts for syntax errors.

-s Take commands from the standard input.

-t Read and execute a single command line. A `\' (backslash) can be used to escape each newline for continuation of the command line onto subsequent input lines.

-v Verbose. Set the verbose predefined variable; command input is echoed after history substitution (but before other substitutions) and before execution.

-V Set verbose before reading .cshrc.

-x Echo. Set the echo variable; echo commands after all substitutions and just before execution.

-X Set echo before reading .cshrc.

Except with the options -c, -i, -s, or -t, the first nonoption argument is taken to be the name of a command or script. It is passed as argument zero, and subsequent arguments are added to the argument list for that command or script.

USAGE

Filename Completion

When enabled by setting the variable filec, an interactive C shell can complete a partially typed filename or user name. When an unambiguous partial filename is followed by an ESC character on the terminal input line, the shell fills in the remaining characters of a matching filename from the working directory.

If a partial filename is followed by the EOF character (usually typed as CTRL-d), the shell lists all filenames that match. It then prompts once again, supplying the incomplete command line typed in so far.

When the last (partial) word begins with a tilde (~), the shell attempts completion with a user name, rather than a file in the working directory.

The terminal bell signals errors or multiple matches; this can be inhibited by setting the variable nobeep. You can exclude files with certain suffixes by listing those suffixes in the variable fignore. If, however, the only possible completion includes a suffix in the list, it is not ignored. fignore does not affect the listing of filenames by the EOF character.

Lexical Structure

The shell splits input lines into words at space and tab

characters, except as noted below. The characters &, |, ;, <, >, (, and) form separate words; if paired, the pairs form single words. These shell metacharacters can be made part of other words, and their special meaning can be suppressed by preceding them with a `\' (backslash). A newline preceded by a \ is equivalent to a space character.

In addition, a string enclosed in matched pairs of single-quotes ('), double-quotes ("), or backquotes (`), forms a partial word; metacharacters in such a string, including any space or tab characters, do not form separate words. Within pairs of backquote (`) or double-quote (") characters, a newline preceded by a `\' (backslash) gives a true newline character. Additional functions of each type of quote are described, below, under Variable Substitution, Command Substitution, and Filename Substitution.

When the shell's input is not a terminal, the character # introduces a comment that continues to the end of the input line. Its special meaning is suppressed when preceded by a \ or enclosed in matching quotes.

Command Line Parsing

A simple command is composed of a sequence of words. The first word (that is not part of an I/O redirection) specifies the command to be executed. A simple command, or a set of simple commands separated by | or |& characters, forms a pipeline. With |, the standard output of the preceding command is redirected to the standard input of the command that follows. With |&, both the standard error and the standard output are redirected through the pipeline.

Pipelines can be separated by semicolons (;), in which case they are executed sequentially. Pipelines that are separated by && or || form conditional sequences in which the execution of pipelines on the right depends upon the success or failure, respectively, of the pipeline on the left.

A pipeline or sequence can be enclosed within parentheses `( )' to form a simple command that can be a component in a pipeline or sequence.

A sequence of pipelines can be executed asynchronously or "in the background" by appending an `&'; rather than waiting for the sequence to finish before issuing a prompt, the shell displays the job number (see Job Control, below) and associated process IDs and prompts immediately.

History Substitution

History substitution allows you to use words from previous command lines in the command line you are typing. This simplifies spelling corrections and the repetition of complicated commands or arguments. Command lines are saved in the history list, the size of which is controlled by the history variable. The most recent command is retained in any case. A history substitution begins with a ! (although you can change this with the histchars variable) and may occur anywhere on the command line; history substitutions do not nest. The ! can be escaped with \ to suppress its special meaning.

Input lines containing history substitutions are echoed on the terminal after being expanded, but before any other substitutions take place or the command gets executed.

Event Designators

An event designator is a reference to a command line entry in the history list.

- `!` Start a history substitution, except when followed by a space character, tab, newline, = or (.
- `!!` Refer to the previous command. By itself, this substitution repeats the previous command.
- `!n` Refer to command line n.
- `!-n` Refer to the current command line minus n.
- `!str` Refer to the most recent command starting with str.
- `!?str?` Refer to the most recent command containing str.
- `!?str? additional` Refer to the most recent command containing str and append additional to that referenced command.
- `!{command} additional` Refer to the most recent command beginning with command and append additional to that referenced command.
- `^previous_word^replacement^` Repeat the previous command line replacing the string previous_word with the string replacement. This is equivalent to the history substitution:

  `!:s/previous_word/replacement/.`

  To re-execute a specific previous command AND make such a substitution, say, re-executing command #6,

  `!:6s/previous_word/replacement/.`

Word Designators

A `:' (colon) separates the event specification from the word designator. It can be omitted if the word designator begins with a ^, $, *, - or %. If the word is to be selected from the previous command, the second ! character can be omitted from the event specification. For instance, !!:1 and !:1 both refer to the first word of the previous command, while !!$ and !$ both refer to the last word in the previous command. Word designators include:

#	The entire command line typed so far.
0	The first input word (command).
n	The n'th argument.
^	The first argument, that is, 1.
$	The last argument.
%	The word matched by (the most recent) ?s search.
x-y	A range of words; -y abbreviates 0-y.
*	All the arguments, or a null value if there is just one word in the event.
x*	Abbreviates x-$.
x-	Like x* but omitting word $.

Modifiers

After the optional word designator, you can add one of the following modifiers, preceded by a :.

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing suffix of the form `.xxx', leaving the basename.
e	Remove all but the suffix, leaving the Extension.
s/l/r/	Substitute r for l.
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change to the first occurrence of a match in each word, by prefixing the above (for example, g&).
p	Print the new command but do not execute it.
q	Quote the substituted words, escaping further substitutions.
x	Like q, but break into words at each space character, tab or newline.

Unless preceded by a g, the modification is applied only to the first string that matches l; an error results if no

string matches.

The left-hand side of substitutions are not regular expressions, but character strings. Any character can be used as the delimiter in place of /. A backslash quotes the delimiter character. The character &, in the right hand side, is replaced by the text from the left-hand-side. The & can be quoted with a backslash. A null l uses the previous string either from a l or from a contextual scan string s from !?s. You can omit the rightmost delimiter if a newline immediately follows r; the rightmost ? in a context scan can similarly be omitted.

Without an event specification, a history reference refers either to the previous command, or to a previous history reference on the command line (if any).

Quick Substitution

^l^r^

This is equivalent to the history substitution: !:s/l/r/.

Aliases

The C shell maintains a list of aliases that you can create, display, and modify using the alias and unalias commands. The shell checks the first word in each command to see if it matches the name of an existing alias. If it does, the command is reprocessed with the alias definition replacing its name; the history substitution mechanism is made available as though that command were the previous input line. This allows history substitutions, escaped with a backslash in the definition, to be replaced with actual command line arguments when the alias is used. If no history substitution is called for, the arguments remain unchanged.
Aliases can be nested. That is, an alias definition can contain the name of another alias. Nested aliases are expanded before any history substitutions is applied. This is useful in pipelines such as

```
alias lm 'ls -l \!* | more'
```

which when called, pipes the output of ls(1) through more(1).

Except for the first word, the name of the alias may not appear in its definition, nor in any alias referred to by its definition. Such loops are detected, and cause an error

message.

I/O Redirection

The following metacharacters indicate that the subsequent word is the name of a file to which the command's standard input, standard output, or standard error is redirected; this word is variable, command, and filename expanded separately from the rest of the command.

`<` Redirect the standard input.

`<<word` Read the standard input, up to a line that is identical with word, and place the resulting lines in a temporary file. Unless word is escaped or quoted, variable and command substitutions are performed on these lines. Then, the pipeline is invoked with the temporary file as its standard input. word is not subjected to variable, filename, or command substitution, and each line is compared to it before any substitutions are performed by the shell.

`>` `>!` `>&` `>&!`
Redirect the standard output to a file. If the file does not exist, it is created. If it does exist, it is overwritten; its previous contents are lost.

When set, the variable noclobber prevents destruction of existing files. It also prevents redirection to terminals and /dev/null, unless one of the ! forms is used. The & forms redirect both standard output and the standard error (diagnostic output) to the file.

`>>` `>>&` `>>!` `>>&!`
Append the standard output. Like >, but places output at the end of the file rather than overwriting it. If noclobber is set, it is an error for the file not to exist, unless one of the ! forms is used. The & forms append both the standard error and standard output to the file.

Variable Substitution

The C shell maintains a set of variables, each of which is

composed of a name and a value. A variable name consists of up to 20 letters and digits, and starts with a letter (the underscore is considered a letter). A variable's value is a space-separated list of zero or more words.

To refer to a variable's value, precede its name with a `$'. Certain references (described below) can be used to select specific words from the value, or to display other information about the variable. Braces can be used to insulate the reference from other characters in an input-line word.

Variable substitution takes place after the input line is analyzed, aliases are resolved, and I/O redirections are applied. Exceptions to this are variable references in I/O redirections (substituted at the time the redirection is made), and backquoted strings (see Command Substitution).

Variable substitution can be suppressed by preceding the $ with a \, except within double-quotes where it always occurs. Variable substitution is suppressed inside of single-quotes. A $ is escaped if followed by a space character, tab or newline.

Variables can be created, displayed, or destroyed using the set and unset commands. Some variables are maintained or used by the shell. For instance, the argv variable contains an image of the shell's argument list. Of the variables used by the shell, a number are toggles; the shell does not care what their value is, only whether they are set or not.

Numerical values can be operated on as numbers (as with the @ built-in command). With numeric operations, an empty value is considered to be zero; the second and subsequent words of multiword values are ignored. For instance, when the verbose variable is set to any value (including an empty value), command input is echoed on the terminal.

Command and filename substitution is subsequently applied to the words that result from the variable substitution, except when suppressed by double-quotes, when noglob is set (suppressing filename substitution), or when the reference is quoted with the :q modifier. Within double-quotes, a reference is expanded to form (a portion of) a quoted string; multiword values are expanded to a string with embedded space characters. When the :q modifier is applied to the reference, it is expanded to a list of space-separated words, each of which is quoted to prevent subse-

quent command or filename substitutions.

Except as noted below, it is an error to refer to a variable that is not set.

$var
${var} These are replaced by words from the value of var, each separated by a space character. If var is an environment variable, its value is returned (but `:' modifiers and the other forms given below are not available).

$var[index]
${var[index]} These select only the indicated words from the value of var. Variable substitution is applied to index, which may consist of (or result in) a either single number, two numbers separated by a `-', or an asterisk. Words are indexed starting from 1; a `*' selects all words. If the first number of a range is omitted (as with $argv[-2]), it defaults to 1. If the last number of a range is omitted (as with $argv[1-]), it defaults to $#var (the word count). It is not an error for a range to be empty if the second argument is omitted (or within range).

$#name
${#name} These give the number of words in the variable.

$0 This substitutes the name of the file from which command input is being read except for setuid shell scripts. An error occurs if the name is not known.

$n
${n} Equivalent to $argv[n].

$* Equivalent to $argv[*].

The modifiers :e, :h, :q, :r, :t, and :x can be applied (see History Substitution), as can :gh, :gt, and :gr. If {} (braces) are used, then the modifiers must appear within the braces. The current implementation allows only one such modifier per expansion.
The following references may not be modified with : modif-

iers.

$?var
${?var}
Substitutes the string 1 if var is set or 0 if it is not set.

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

$$ Substitute the 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 C shell script.

Command and Filename Substitutions

Command and filename substitutions are applied selectively to the arguments of built-in commands. Portions of expressions that are not evaluated are not expanded. For non-built-in commands, filename expansion of the command name is done separately from that of the argument list; expansion occurs in a subshell, after I/O redirection is performed.

Command Substitution

A command enclosed by backquotes (`...`) is performed by a subshell. Its standard output is broken into separate words at each space character, tab and newline; null words are discarded. This text replaces the backquoted string on the current command line. Within double-quotes, only newline characters force new words; space and tab characters are preserved. However, a final newline is ignored. It is therefore possible for a command substitution to yield a partial word.

Filename Substitution

Unquoted words containing any of the characters *, ?, [or {, or that begin with ~, are expanded (also known as globbing) to an alphabetically sorted list of filenames, as follows:

* Match any (zero or more) characters.

? Match any single character.

[...] Match any single character in the enclosed list(s) or range(s). A list is a string of

characters. A range is two characters separated by a dash (-), and includes all the characters in between in the ASCII collating sequence (see ascii(5)).

{ str, str, ... }
Expand to each string (or filename-matching pattern) in the comma-separated list. Unlike the pattern-matching expressions above, the expansion of this construct is not sorted. For instance, {b,a} expands to `b' `a', (not `a' `b'). As special cases, the characters { and }, along with the string {}, are passed undisturbed.

~[user] Your home directory, as indicated by the value of the variable home, or that of user, as indicated by the password entry for user.

Only the patterns *, ? and [...] imply pattern matching; an error results if no filename matches a pattern that contains them. The `.' (dot character), when it is the first character in a filename or pathname component, must be matched explicitly. The / (slash) must also be matched explicitly.

Expressions and Operators

A number of C shell built-in commands accept expressions, in which the operators are similar to those of C and have the same precedence. These expressions typically appear in the @, exit, if, set and while commands, and are often used to regulate the flow of control for executing commands. Components of an expression are separated by white space.

Null or missing values are considered 0. The result of all expressions is a string, which may represent decimal numbers.

The following C shell operators are grouped in order of precedence:

(...)	grouping
~	one's complement
!	logical negation
* / %	multiplication, division, remainder

(These are right associative, which can lead to unexpected results. Group combinations explicitly with

	parentheses.)
+ -	addition, subtraction (also right associative)
<< >>	bitwise shift left, bitwise shift right
< > <= >=	less than, greater than, less than or equal to, greater than or equal to
== != =~ !~	equal to, not equal to, filename-substitution pattern match (described below), filename-substitution pattern mismatch
&	bitwise AND
^	bitwise XOR (exclusive or)
|	bitwise inclusive OR
&&	logical AND
||	logical OR

The operators: ==, !=, =~, and !~ compare their arguments as strings; other operators use numbers. The operators =~ and !~ each check whether or not a string to the left matches a filename substitution pattern on the right. This reduces the need for switch statements when pattern-matching between strings is all that is required.

Also available are file inquiries:

-r filename	Return true, or 1 if the user has read access. Otherwise it returns false, or 0.
-w filename	True if the user has write access.
-x filename	True if the user has execute permission (or search permission on a directory).
-e filename	True if filename exists.
-o filename	True if the user owns filename.
-z filename	True if filename is of zero length (empty).
-f filename	True if filename is a plain file.
-d filename	True if filename is a directory.

If filename does not exist or is inaccessible, then all inquiries return false.

An inquiry as to the success of a command is also available:

{ command }	If command runs successfully, the expression evaluates to true, 1. Otherwise, it evaluates to false, 0. (Note: Conversely, command itself typically returns

0 when it runs successfully, or some other value if it encounters a problem. If you want to get at the status directly, use the value of the status variable rather than this expression).

Control Flow

The shell contains a number of commands to regulate the flow of control in scripts and within limits, from the terminal. These commands operate by forcing the shell either to reread input (to loop), or to skip input under certain conditions (to branch).

Each occurrence of a foreach, switch, while, if...then and else built-in command must appear as the first word on its own input line.

If the shell's input is not seekable and a loop is being read, that input is buffered. The shell performs seeks within the internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto commands will succeed on nonseekable inputs.)

Command Execution

If the command is a C shell built-in command, the shell executes it directly. Otherwise, the shell searches for a file by that name with execute access. If the command name contains a /, the shell takes it as a pathname, and searches for it. If the command name does not contain a /, the shell attempts to resolve it to a pathname, searching each directory in the path variable for the command. To speed the search, the shell uses its hash table (see the rehash built-in command) to eliminate directories that have no applicable files. This hashing can be disabled with the -c or -t, options, or the unhash built-in command.

As a special case, if there is no / in the name of the script and there is an alias for the word shell, the expansion of the shell alias is prepended (without modification) to the command line. The system attempts to execute the first word of this special (late-occurring) alias, which should be a full pathname. Remaining words of the alias's definition, along with the text of the input line, are treated as arguments.

When a pathname is found that has proper execute permissions, the shell forks a new process and passes it, along with its arguments, to the kernel using the execve() system

call (see exec(2)). The kernel then attempts to overlay the new process with the desired program. If the file is an executable binary (in a.out(4) format) the kernel succeeds and begins executing the new process. If the file is a text file and the first line begins with #!, the next word is taken to be the pathname of a shell (or command) to interpret that script. Subsequent words on the first line are taken as options for that shell. The kernel invokes (overlays) the indicated shell, using the name of the script as an argument.

If neither of the above conditions holds, the kernel cannot overlay the file and the execve() call fails (see exec(2)); the C shell then attempts to execute the file by spawning a new shell, as follows:

+ If the first character of the file is a #, a C shell is invoked.
+ Otherwise, a Bourne shell is invoked.

Signal Handling

The shell normally ignores QUIT signals. Background jobs are immune to signals generated from the keyboard, including hangups (HUP). Other signals have the values that the C shell inherited from its environment. The shell's handling of interrupt and terminate signals within scripts can be controlled by the onintr built-in command. Login shells catch the TERM signal; otherwise, this signal is passed on to child processes. In no case are interrupts allowed when a login shell is reading the .logout file.

Job Control

The shell associates a numbered job with each command sequence to keep track of those commands that are running in the background or have been stopped with TSTP signals (typically CTRL-z). When a command or command sequence (semicolon separated list) is started in the background using the & metacharacter, the shell displays a line with the job number in brackets and a list of associated process numbers:

```
[1] 1234
```

To see the current list of jobs, use the jobs built-in command. The job most recently stopped (or put into the background if none are stopped) is referred to as the current job and is indicated with a `+'. The previous job is indicated with a `-'; when the current job is terminated or moved to the foreground, this job takes its place (becomes

the new current job).

To manipulate jobs, refer to the bg, fg, kill, stop, and % built-in commands.

A reference to a job begins with a `%'. By itself, the percent-sign refers to the current job.

% %+ %%	The current job.
%-	The previous job.
%j	Refer to job j as in: `kill -9 %j'. j can be a job number, or a string that uniquely specifies the command line by which it was started; `fg %vi' might bring a stopped vi job to the foreground, for instance.
%?string	Specify the job for which the command line uniquely contains string.

A job running in the background stops when it attempts to read from the terminal. Background jobs can normally produce output, but this can be suppressed using the `stty tostop' command.

Status Reporting

While running interactively, the shell tracks the status of each job and reports whenever the job finishes or becomes blocked. It normally displays a message to this effect as it issues a prompt, in order to avoid disturbing the appearance of your input. When set, the notify variable indicates that the shell is to report status changes immediately. By default, the notify command marks the current process; after starting a background job, type notify to mark it.

Built-In Commands

Built-in commands are executed within the C shell. If a built-in command occurs as any component of a pipeline except the last, it is executed in a subshell.

: Null command. This command is interpreted, but performs no action.

alias [name [def]]
Assign def to the alias name. def is a list of words that may contain escaped history-substitution metasyntax. name is not allowed to be alias or unalias. If def is omitted, the current definition for the alias name is

displayed. If both name and def are omitted, all aliases are displayed with their definitions.

bg [%job ...]
Run the current or specified jobs in the background.

break Resume execution after the end of the nearest enclosing foreach or while loop. The remaining commands on the current line are executed. This allows multilevel breaks to be written as a list of break commands, all on one line.

breaksw Break from a switch, resuming after the endsw.

case label:
A label in a switch statement.

cd [dir]
chdir [dir]
Change the shell's working directory to directory dir. If no argument is given, change to the home directory of the user. If dir is a relative pathname not found in the current directory, check for it in those directories listed in the cdpath variable. If dir is the name of a shell variable whose value starts with a /, change to the directory named by that value.

continue Continue execution of the next iteration of the nearest enclosing while or foreach loop.

default: Labels the default case in a switch statement. The default should come after all case labels. Any remaining commands on the command line are first executed.

dirs [-l]
Print the directory stack, most recent to the left; the first directory shown is the current directory. With the -l argument, produce an unabbreviated printout; use of the ~ notation is suppressed.

echo [-n] list
The words in list are written to the shell's standard output, separated by space characters. The

output is terminated with a newline unless the -n option is used.
csh will, by default, invoke its built-in echo, if echo is called without the full pathname of a Unix command, regardless of the configuration of your PATH (see echo(1)).

eval argument...
Reads the arguments as input to the shell and executes the resulting command(s). This is usually used to execute commands generated as the result of command or variable substitution. See tset(1B) for an example of how to use eval.

exec command
Execute command in place of the current shell, which terminates.

exit [(expr)]
The calling shell or shell script exits, either with the value of the status variable or with the value specified by the expression expr.

fg [%job]
Bring the current or specified job into the foreground.

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

The built-in command continue may be used to terminate the execution of the current iteration of the loop and the built-in command break may be used to terminate execution of the foreach command. When this command is read from the terminal, the loop is read once prompting with ? before any statements in the loop are executed.

glob wordlist
Perform filename expansion on wordlist. Like echo, but no \ escapes are recognized. Words are delimited by NULL characters in the output.

goto label
The specified label is a filename and a command expanded to yield a label. The shell rewinds its input as much as possible and searches for a line of the form label: possibly preceded by space or tab characters. Execution continues after the indicated line. It is an error to jump to a label that occurs between a while or for built-in command and its corresponding end.

hashstat Print a statistics line indicating how effective the internal hash table for the path variable has been at locating commands (and avoiding execs). An exec is attempted for each component of the path where the hash function indicates a possible hit and in each component that does not begin with a `/'. These statistics only reflect the effectiveness of the path variable, not the cdpath variable.

history [-hr] [n]
Display the history list; if n is given, display only the n most recent events.

-r Reverse the order of printout to be most recent first rather than oldest first.

-h Display the history list without leading numbers. This is used to produce files suitable for sourcing using the -h option to source.

if (expr) command
If the specified expression evaluates to true, 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. Note: I/O redirection occurs even if expr is false, when command is not executed (this is a bug).

if (expr) then
...
else if (expr2) then
...

else
...
endif If expr is true, commands up to the first else are executed. Otherwise, if expr2 is true, the commands between the else if and the second else are executed. Otherwise, commands between the else and the endif are executed. Any number of else if pairs are allowed, but only one else. Only one endif is needed, but it is required. The words else and endif must be the first nonwhite characters on a line. The if must appear alone on its input line or after an else.

jobs[-l] List the active jobs under job control.

-l List process IDs, in addition to the normal information.

kill [-sig] [pid] [%job] ...
kill -l Send the TERM (terminate) signal, by default, or the signal specified, to the specified process ID, the job indicated, or the current job. Signals are either given by number or by name. There is no default. Typing kill does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process is sent a CONT (continue) signal as well.

-l List the signal names that can be sent.

limit [-h] [resource [max-use]]
Limit the consumption by the current process or any process it spawns, each not to exceed max-use on the specified resource. If max-use is omitted, print the current limit; if resource is omitted, display all limits. (Run the sysdef(1M) command to obtain the maximum possible limits for your system. The values reported are in hexadecimal, but can be translated into decimal numbers using the bc(1) command).

-h Use hard limits instead of the current limits. Hard limits impose a ceiling on the values of the current limits. Only the privileged user may raise the hard limits.

resource is one of:

cputime	Maximum CPU seconds per process.
filesize	Largest single file allowed; limited to the size of the filesystem. (see df(1M)).
datasize (heapsize)	Maximum data size (including stack) for the process. This is the size of your virtual memory See swap(1M).
stacksize	Maximum stack size for the process. See swap(1M).
coredumpsize	Maximum size of a core dump (file). This limited to the size of the filesystem.
descriptors	Maximum number of file descriptors. Run sysdef().
memorysize	Maximum size of virtual memory.

max-use is a number, with an optional scaling factor, as follows:

nh	Hours (for cputime).
nk	n kilobytes. This is the default for all but cputime.
nm	n megabytes or minutes (for cputime).
mm:ss	Minutes and seconds (for cputime).

Example of limit: to limit the size of a core file dump to 0 Megabytes, type the following:

```
limit coredumpsize 0M
```

login [username| -p]

Terminate a login shell and invoke login(1). The .logout file is not processed. If username is omitted, login prompts for the name of a user.

-p Preserve the current environment (variables).

logout Terminate a login shell.

nice [+n | -n] [command]

Increment the process priority value for the shell or for command by n. The higher the priority value, the lower the priority of a process, and

the slower it runs. When given, command is always run in a subshell, and the restrictions placed on commands in simple if commands apply. If command is omitted, nice increments the value for the current shell. If no increment is specified, nice sets the process priority value to 4. The range of process priority values is from -20 to 20. Values of n outside this range set the value to the lower, or to the higher boundary, respectively.

+n Increment the process priority value by n.

-n Decrement by n. This argument can be used only by the privileged user.

nohup [command]
Run command with HUPs ignored. With no arguments, ignore HUPs throughout the remainder of a script. When given, command is always run in a subshell, and the restrictions placed on commands in simple if statements apply. All processes detached with & are effectively nohup'd.

notify [%job] ...
Notify the user asynchronously when the status of the current job or specified jobs changes.

onintr [-| label]
Control the action of the shell on interrupts. With no arguments, onintr restores the default action of the shell on interrupts. (The shell terminates shell scripts and returns to the terminal command input level). With the - argument, the shell ignores all interrupts. With a label argument, the shell executes a goto label when an interrupt is received or a child process terminates because it was interrupted.

popd [+n] Pop the directory stack and cd to the new top directory. The elements of the directory stack are numbered from 0 starting at the top.

+n Discard the n'th entry in the stack.

pushd [+n | dir]
Push a directory onto the directory stack. With no arguments, exchange the top two elements.

+n Rotate the n'th entry to the top of the stack and cd to it.

dir Push the current working directory onto the stack and change to dir.

rehash Recompute the internal hash table of the contents of directories listed in the path variable to account for new commands added. Recompute the internal hash table of the contents of directories listed in the cdpath variable to account for new directories added.

repeat count command
Repeat command count times. command is subject to the same restrictions as with the one-line if statement.

set [var [= value]]
set var[n] = word
With no arguments, set displays the values of all shell variables. Multiword values are displayed as a parenthesized list. With the var argument alone, set assigns an empty (null) value to the variable var. With arguments of the form var = value set assigns value to var, where value is one of:

word A single word (or quoted string).
(wordlist) A space-separated list of words enclosed in parentheses.

Values are command and filename expanded before being assigned. The form set var[n] = word replaces the n'th word in a multiword value with word.

setenv [VAR [word]]

With no arguments, setenv displays all *ENVIRONMENT*
variables. With the VAR argument, setenv sets the environment variable VAR to have an empty (null) value. (By convention, environment variables are normally given upper-case names.) With both VAR

and word arguments, setenv sets the *ENVIRONMENT*

variable NAME to the value word, which must be either a single word or a quoted string. The most commonly used environment variables, USER, TERM, and PATH, are automatically imported to and exported from the csh variables user, term, and path; there is no need to use setenv for these.

In addition, the shell sets the PWD *ENVIRONMENT*

variable from the csh variable cwd whenever the latter changes.

The environment variables LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC, and LC_MONETARY take immediate effect when changed within the C shell.

If any of the LC_* variables (LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC, and LC_MONETARY) (see environ(5)) are not set in the environment, the operational behavior of csh for each corresponding locale category is determined by the value of the LANG environment variable. If LC_ALL is set, its contents are used to override both the LANG and the other LC_* variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how csh behaves.

LC_CTYPE

Determines how csh handles characters. When LC_CTYPE is set to a valid value, csh can display and handle text and filenames containing valid characters for that locale.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S./English).

LC_NUMERIC
Determines the value of the radix character (decimal point (".") in the "C" locale) and thousand separator (empty string ("") in the "C" locale).

shift [variable]
The components of argv, or variable, if supplied, are shifted to the left, discarding the first component. It is an error for the variable not to be set or to have a null value.

source [-h] name
Reads commands from name. source commands may be nested, but if they are nested too deeply the shell may run out of file descriptors. An error in a sourced file at any level terminates all nested source commands.

-h Place commands from the file name on the history list without executing them.

stop %jobid ...
Stop the current or specified background job.

stop pid ...
Stop the specified process, pid. (see ps(1)).

suspend Stop the shell in its tracks, much as if it had been sent a stop signal with ^Z. This is most often used to stop shells started by su.

switch (string)
case label:
...
breaksw
...
default:
...
breaksw
endsw
Each label is successively matched, against the specified string, which is first command and filename expanded. The file metacharacters *, ? and [...] may be used in the case labels, which are variable expanded. If none of the labels match before a "default" label is found, execution begins after the default label. Each case state-

ment and the default statement must appear at the beginning of a line. The command breaksw continues execution after the endsw. Otherwise control falls through subsequent case and default statements as with C. If no label matches and there is no default, execution continues after the endsw.

time [command]
With no argument, print a summary of time used by this C shell and its children. With an optional command, execute command and print a summary of the time it uses.

As of this writing, the time built-in command does NOT compute the last 6 fields of output, rendering the output to erroneously report the value "0" for these fields.

```
example %time ls -R
9.0u 11.0s 3:32 10% 0+0k 0+0io 0pf+0w
```

(See below the "Environment Variables and Predefined Shell Variables" sub-section on the time variable.)

umask [value]
Display the file creation mask. With value, set the file creation mask. With value given in octal, the user can turn-off any bits, but cannot turn-on bits to allow new permissions. Common values include 077, restricting all permissions from everyone else; 002, giving complete access to the group, and read (and directory search) access to others; or 022, giving read (and directory search) but not write permission to the group and others.

unalias pattern
Discard aliases that match (filename substitution) pattern. All aliases are removed by `unalias *'.

unhash Disable the internal hash tables for the path and cdpath variables.

unlimit [-h] [resource]
Remove a limitation on resource. If no resource is specified, then all resource limitations are

removed. See the description of the limit command for the list of resource names.

-h Remove corresponding hard limits. Only the privileged user may do this.

unset pattern
Remove variables whose names match (filename substitution) pattern. All variables are removed by `unset *'; this has noticeably distasteful side effects.

unsetenv variable
Remove variable from the environment. As with unset, pattern matching is not performed.

wait Wait for background jobs to finish (or for an interrupt) before prompting.

while (expr)
...
end While expr is true (evaluates to nonzero), repeat commands between the while and the matching end statement. break and continue may be used to terminate or continue the loop prematurely. The while and end must appear alone on their input lines. If the shell's input is a terminal, it prompts for commands with a question-mark until the end command is entered and then performs the commands in the loop.

%[job] [&]
Bring the current or indicated job to the foreground. With the ampersand, continue running job in the background.

@ [var =expr]
@ [var[n] =expr]
With no arguments, display the values for all shell variables. With arguments, set the variable var, or the n'th word in the value of var, to the value that expr evaluates to. (If [n] is supplied, both var and its n'th component must already exist.)

If the expression contains the characters >, <, &, or |, then at least this part of expr must be placed within parentheses.

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

Special postfix operators, ++ and --, increment or decrement name, respectively.

Environment Variables and Predefined Shell Variables

Unlike the Bourne shell, the C shell maintains a distinction between environment variables, which are automatically exported to processes it invokes, and shell variables, which are not. Both types of variables are treated similarly under variable substitution. The shell sets the variables argv, cwd, home, path, prompt, shell, and status upon initialization. The shell copies the environment variable USER into the shell variable user, TERM into term, and HOME into home, and copies each back into the respective environment variable whenever the shell variables are reset. PATH and path are similarly handled. You need only set path once in the .cshrc or .login file. The environment variable PWD is set from cwd whenever the latter changes. The following shell variables have predefined meanings:

argv — Argument list. Contains the list of command line arguments supplied to the current invocation of the shell. This variable determines the value of the positional parameters $1, $2, and so on.

cdpath — Contains a list of directories to be searched by the cd, chdir, and popd commands, if the directory argument each accepts is not a subdirectory of the current directory.

cwd — The full pathname of the current directory.

echo — Echo commands (after substitutions) just before execution.

fignore — A list of filename suffixes to ignore when attempting filename completion. Typically the single word `.o'.

filec — Enable filename completion, in which case the CTRL-d character EOT and the ESC character have special significance when typed in at the end of a terminal input line:

EOT Print a list of all filenames that start with the preceding string.
ESC Replace the preceding string with the longest unambiguous extension.

hardpaths — If set, pathnames in the directory stack are resolved to contain no symbolic-link components.

histchars — A two-character string. The first character replaces ! as the history-substitution character. The second replaces the carat (^) for quick substitutions.

history — The number of lines saved in the history list. A very large number may use up all of the C shell's memory. If not set, the C shell saves only the most recent command.

home — The user's home directory. The filename expansion of ~ refers to the value of this variable.

ignoreeof — If set, the shell ignores EOF from terminals. This protects against accidentally killing a C shell by typing a CTRL-d.

mail — A list of files where the C shell checks for mail. If the first word of the value is a number, it specifies a mail checking interval in seconds (default 5 minutes).

nobeep — Suppress the bell during command completion when asking the C shell to extend an ambiguous filename.

noclobber — Restrict output redirection so that existing files are not destroyed by accident. > redirections can only be made to new

files. >> redirections can only be made to existing files.

noglob — Inhibit filename substitution. This is most useful in shell scripts once filenames (if any) are obtained and no further expansion is desired.

nonomatch — Returns the filename substitution pattern, rather than an error, if the pattern is not matched. Malformed patterns still result in errors.

notify — If set, the shell notifies you immediately as jobs are completed, rather than waiting until just before issuing a prompt.

path — The list of directories in which to search for commands. path is initialized from the environment variable PATH, which the C shell updates whenever path changes. A null word specifies the current directory. The default is typically (/usr/bin .). If path becomes unset only full pathnames will execute. An interactive C shell will normally hash the contents of the directories listed after reading .cshrc, and whenever path is reset. If new commands are added, use the rehash command to update the table.

prompt — The string an interactive C shell prompts with. Noninteractive shells leave the prompt variable unset. Aliases and other commands in the .cshrc file that are only useful interactively, can be placed after the following test: `if ($?prompt == 0) exit', to reduce startup time for noninteractive shells. A ! in the prompt string is replaced by the current event number. The default prompt is hostname% for mere mortals, or hostname# for the privileged user.

The setting of $prompt has three meanings:

$prompt not set -- non-interactive

shell, test $?prompt.

$prompt set but == "" -- .cshrc called by the which(1) command.

$prompt set and != "" -- normal interactive shell.

savehist — The number of lines from the history list that are saved in ~/.history when the user logs out. Large values for savehist slow down the C shell during startup.

shell — The file in which the C shell resides. This is used in forking shells to interpret files that have execute bits set, but that are not executable by the system.

status — The status returned by the most recent command. If that command terminated abnormally, 0200 is added to the status. Built-in commands that fail return exit status 1; all other built-in commands set status to 0.

time — Control automatic timing of commands. Can be supplied with one or two values. The first is the reporting threshold in CPU seconds. The second is a string of tags and text indicating which resources to report on. A tag is a percent sign (%) followed by a single upper-case letter (unrecognized tags print as text):

- %D Average amount of unshared data space used in Kilobytes.
- %E Elapsed (wallclock) time for the command.
- %F Page faults.
- %I Number of block input operations.
- %K Average amount of unshared stack space used in Kilobytes.
- %M Maximum real memory used during execution of the process.
- %O Number of block output operations.

%P Total CPU time - U (user) plus S (system) - as a percentage of E (elapsed) time.

%S Number of seconds of CPU time consumed by the kernel on behalf of the user's process.

%U Number of seconds of CPU time devoted to the user's process.

%W Number of swaps.

%X Average amount of shared memory used in Kilobytes.

The default summary display outputs from the %U, %S, %E, %P, %X, %D, %I, %O, %F, and %W tags, in that order.

verbose — Display each command after history substitution takes place.

Large File Behavior

See largefile(5) for the description of the behavior of csh when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

FILES

~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by login shells after .cshrc at login.
~/.logout	Read by login shells at logout.
~/.history	Saved history for use at next login.
/usr/bin/sh	The Bourne shell, for shell scripts not starting with a `#'.
/tmp/sh*	Temporary file for `<<'.
/etc/passwd	Source of home directories for `~name'.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled

SEE ALSO

bc(1), echo(1), login(1), ls(1), more(1), ps(1), sh(1), shell_builtins(1), tset(1B), which(1), df(1M), swap(1M), sysdef(1M), access(2), exec(2), fork(2), pipe(2), a.out(4), environ(4), ascii(5), attributes(5), environ(5), large-file(5), termio(7I)

DIAGNOSTICS

You have stopped jobs.

You attempted to exit the C shell with stopped jobs under job control. An immediate second attempt to exit will succeed, terminating the stopped jobs.

WARNINGS

The use of setuid shell scripts is strongly discouraged.

NOTES

Words can be no longer than 1024 bytes. The system limits argument lists to 1,048,576 bytes. However, the maximum number of arguments to a command for which filename expansion applies is 1706. Command substitutions may expand to no more characters than are allowed in the argument list. To detect looping, the shell restricts the number of alias substitutions on a single line to 20.

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

Shell built-in functions are not stoppable/restartable. Command sequences of the form
a ; b ; c are also not handled gracefully when stopping is attempted. If you suspend b, the shell never executes c. This is especially noticeable if the expansion results from an alias. It can be avoided by placing the sequence in parentheses to force it into a subshell.
Control over terminal output after processes are started is primitive; use the Sun Window system if you need better output control.

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

Control structures should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. There are two problems with : modifier usage on variable substitutions: not all of the modifiers are available, and only one modifier per substitution is allowed.

The g (global) flag in history substitutions applies only to the first match in each word, rather than all matches in all words. The common text editors consistently do the latter when given the g flag in a substitution command.

Quoting conventions are confusing. Overriding the escape character to force variable substitutions within double quotes is counterintuitive and inconsistent with the Bourne shell.

Symbolic links can fool the shell. Setting the hardpaths variable alleviates this.

It is up to the user to manually remove all duplicate pathnames accrued from using built-in commands as

```
set path = pathnames
```

or

```
setenv PATH pathnames
```

more than once. These often occur because a shell script or a .cshrc file does something like `set path=(/usr/local /usr/hosts $path)' to ensure that the named directories are in the pathname list.

The only way to direct the standard output and standard error separately is by invoking a subshell, as follows:

```
example% (command > outfile) >& errorfile
```

Although robust enough for general use, adventures into the esoteric periphery of the C shell may reveal unexpected quirks.

If you start csh as a login shell and you do not have a .login in your home directory, then the csh reads in the /etc/.login.

When the shell executes a shell script that attempts to execute a non-existent command interpreter, the shell returns an erroneous diagnostic message that the shell script file does not exist.

BUGS

As of this writing, the time built-in command does NOT compute the last 6 fields of output, rendering the output to erroneously report the value "0" for these fields.

```
example %time ls -R
9.0u 11.0s 3:32 10% 0+0k 0+0io 0pf+0w
```

cut

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

SYNOPSIS

```
cut -b list [ -n ] [ file ... ]
cut -c list [ file ... ]
cut -f list [ -d delim ] [ -s ] [ file ...  ]
```

DESCRIPTION

Use cut to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by list can be fixed length, that is, 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.

Either the -b, -c, or -f option must be specified.

Use grep(1) to make horizontal ``cuts'' (by context) through a file, or paste(1) to put files together column-wise (that is, horizontally). To reorder columns in a table, use cut and paste.

OPTIONS

list — A comma-separated or blank-character-separated list of integer field numbers (in increasing order), with optional - to indicate ranges (for instance, 1,4,7; 1-3,8; -5,10 (short for 1-5,10); or 3- (short for third through last field)).

-b list — The list following -b specifies byte positions (for instance, -b1-72 would pass the first 72 bytes of each line). When -b and -n are used together, list is adjusted so that no multi-byte character is split. If -b is used, the input line should contain 1023 bytes or less.

-c list — The list following -c specifies character positions (for instance, -c1-72 would pass the first 72 characters of each line).

-d delim 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. delim can be a multi-byte character.

-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 instance, -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. If -f is used, the input line should contain 1023 characters or less.

-n Do not split characters. When -b list and -n are used together, list is adjusted so that no multi-byte character is split.

-s Suppresses lines with no delimiter characters in case of -f option. Unless specified, lines with no delimiters will be passed through untouched.

OPERANDS

The following operands are supported:

file A path name of an input file. If no file operands are specified, or if a file operand is -, the standard input will be used.

USAGE

See largefile(5) for the description of the behavior of cut when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

A mapping of user IDs to names follows:

```
example% cut -d: -f1,5 /etc/passwd
```

To set name to current login name:

```
example$ name=`who am i | cut -f1 -d' '`
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of cut: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 All input files were output successfully.

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled

SEE ALSO

grep(1), paste(1), attributes(5), environ(5), largefile(5)

DIAGNOSTICS

cut: -n may only be used with -b

cut: -d may only be used with -f

cut: -s may only be used with -f

cut: cannot open <file>
Either file cannot be read or does not exist. If multiple files are present, processing continues.

cut: no delimiter specified
Missing delim on -d option.

cut: invalid delimiter

cut: no list specified
Missing list on -b, -c, or -f, option.

cut: invalid range specifier

```
cut: too many ranges specified
cut: range must be increasing
cut: invalid character in range
cut: internal error processing input
cut: invalid multibyte character
cut: unable to allocate enough memory
```

date

date - write the date and time

SYNOPSIS

```
/usr/bin/date [-u] [+format]
/usr/bin/date [-a [-]sss.fff]
/usr/bin/date [-u] [[mmdd]HHMM | mmddHHMM[cc]yy][.SS]

/usr/xpg4/bin/date [-u] [+format]
/usr/xpg4/bin/date [-a [-]sss.fff]
/usr/xpg4/bin/date [-u] [[mmdd]HHMM | mmddHHMM[cc]yy][.SS]
```

DESCRIPTION

The date utility writes the date and time to standard output or attempts to set the system date and time. By default, the current date and time will be written.

Specifications of native language translations of month and weekday names are supported. The month and weekday names used for a language are based on the locale specified by the environment variable LC_TIME; see environ(5).

The following is the default form for the "C" locale:

```
%a %b %e %T %Z %Y
```

for example,

```
Fri Dec 23 10:10:42 EST 1988
```

OPTIONS

The following options are supported:

-a [-]sss.fff
: Slowly adjust the time by sss.fff seconds (fff represents fractions of a second). This adjustment can be positive or negative. The system's clock will be sped up or slowed down until it has drifted by the number of seconds specified.

-u
: Display (or set) the date in Greenwich Mean Time (GMT-universal time), bypassing the normal

conversion to (or from) local time.

OPERANDS

The following operands are supported:

+format

If the argument begins with +, the output of date is the result of passing format and the current time to strftime(). date uses the conversion specifications listed on the strftime(3C) manual page, with the conversion specification for %C determined by whether /usr/bin/date or /usr/xpg4/bin/date is used:

/usr/bin/date
: Locale's date and time representation. This is the default output for date.

/usr/xpg4/bin/date
: Century (a year divided by 100 and truncated to an integer) as a decimal number [00-99].

The string is always terminated with a NEWLINE. An argument containing blanks must be quoted; see the EXAMPLES section.

mm	Month number
dd	Day number in the month
HH	Hour number (24 hour system)
MM	Minute number
SS	Second number
cc	Century minus one
yy	Last 2 digits of the year number

The month, day, year, and century may be omitted; the current values are applied as defaults. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 a.m. The current year is the default because no year is supplied. The system operates in GMT. date takes care of the conversion to and from local standard and daylight time. Only the super-user may change the date. After successfully setting the date and time, date displays the new date according

to the default format. The date command uses TZ to determine the correct time zone information; see environ(5).

EXAMPLES

The command

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

generates as output:

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

The command

```
example# date 1234.56
```

sets the current time to 12:34:56.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of date: LC_CTYPE, LC_TIME, LC_MESSAGES, and NLSPATH.

TZ — Determine the timezone in which the time and date are written, unless the -u option is specified. If the TZ variable is not set and the -u is not specified, the system default timezone is used.

EXIT STATUS

The following exit values are returned:

0 — Successful completion.

>0 — An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/date

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	enabled

/usr/xpg4/bin/date

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	enabled

SEE ALSO

strftime(3C), attributes(5), environ(5), xpg4(5)

DIAGNOSTICS

no permission	You are not the super-user and you tried to change the date.
bad conversion	The date set is syntactically incorrect.

NOTES

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

df

df - report number of free disk blocks and files

SYNOPSIS

```
/usr/bin/df [ -F FSType ] [ -abegklntV ]
     [ -o FSType-specific_options ]
     [ directory | block_device | resource ...  ]

/usr/xpg4/bin/df [ -F FSType ] [ -abegklnPtV ]
     [ -o FSType-specific_options ]
     [ directory | block_device | resource ...  ]
```

DESCRIPTION

The df command displays the amount of disk space occupied by mounted or unmounted file systems, directories, or mounted resources, the amount of used and available space, and how much of the file system's total capacity has been used.

directory represents a valid directory name. If directory is specified, df reports on the file system that contains directory. block_device represents a block special device (for example, /dev/dsk/c1d0s7); if block_device is specified, the corresponding file system need not be mounted. resource is an NFS resource name.

Used without operands or options, df reports on all mounted file systems.

OPTIONS

The following options are supported for both /usr/bin/df and /usr/xpg4/bin/df:

-a Report on all filesystems including ones whose entries in /etc/mnttab (see mnttab(4)) have the ignore option set.

-b Print the total number of kilobytes free.

-e Print only the number of files free.

-F FSType Specify the FSType on which to operate. This is only needed if the file system is

unmounted. The FSType should be specified here or be determinable from /etc/vfstab (see vfstab(4)) have the by matching the directory, block_device, or resource with an entry in the table, or by consulting /etc/default/fs. See default_fs(4).

-g Print the entire statvfs(2) structure. This option is used only for mounted file systems. It cannot be used with the -o option. This option will override the -b, -e, -k, -n, -P, and -t options.

-k Print the allocation in kbytes. The output consists of one line of information for each specified file system. This information includes the file system name, the total space allocated in the file system, the amount of space allocated to existing files, the total amount of space available for the creation of new files by unpriviledged users, and the percentage of normally available space that is currently allocated to all files on the file system. This option will override the -b, -e, -n, and -t options.

-l Report on local file systems only. This option is used only for mounted file systems. It cannot be used with the -o option.

-n Print only the FSType name. Invoked with no operands, this option prints a list of mounted file system types. This option is used only for mounted file systems. It cannot be used with the -o option.

-o FSType-specific_options
Specify FSType-specific options. These options are comma-separated, with no intervening spaces. See the manual page for the FSType-specific command for details.

-t Print full listings with totals. This option will override the -b, -e, and -n options.

-V Echo the complete set of file system specific command lines, but do not execute them. The

command line is generated by using the options and operands provided by the user and adding to them information derived from /etc/mnttab, /etc/vfstab, or /etc/default/fs. This option may be used to verify and validate the command line.

/usr/xpg4/bin/df

The following option is supported for /usr/xpg4/bin/df only:

-P Same as -k except in 512-byte units.

OPERANDS

The following operands are supported:

directory represents a valid directory name. df reports on the file system that contains directory.

block_device represents a block special device (for example, /dev/dsk/c1d0s7); the corresponding file system need not be mounted.

resource represents an NFS resource name.

USAGE

See largefile(5) for the description of the behavior of df when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

The following example writes portable information about the /usr file system:

```
example% /usr/xpg4/bin/df -P /usr
```

Assuming that /usr/src is part of the /usr file system, the following will do the same as the previous example:

```
example% /usr/xpg4/bin/df -P /usr/src
```

ENVIRONMENT

SYSV3 This variable is used to: override the default behavior of df, provide compatibility with INTERACTIVE UNIX System and SCO UNIX

installation scripts, and thus should not be used in new scripts. (It is provided for compatibility only.)

When set, any header which normally displays "files" will now display "nodes". See environ(5) for descriptions of the following environment variables that affect the execution of df: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

FILES

/dev/dsk/* disk devices

/etc/default/fs
default local file system type. Default values can be set for the following flags in /etc/default/fs. For example: LOCAL=ufs
LOCAL: The default partition for a command if no FSType is specified.

/etc/mnttab mount table

/etc/vfstab list of default parameters for each file system

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/df

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

/usr/xpg4/bin/df

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4

SEE ALSO

find(1), mount(1M), statvfs(2), default_fs(4), mnttab(4), vfstab(4), attributes(5), environ(5), largefile(5), xpg4(5)

Manual pages for the FSType-specific modules of df.

NOTES

The -F option is intended for use with unmounted file systems.

This command may not be supported for all FSTypes.

diff

diff - display line-by-line differences between pairs of text files

SYNOPSIS

```
diff [ -bitw ] [ -c | -e | -f | -h | -n ] file1 file2
diff [ -bitw ] [ -C number ] file1 file2
diff [ -bitw ] [ -D string ] file1 file2
diff [ -bitw ] [ -c | -e | -f | -h | -n ] [ -l ] [ -r ]
     [ -s ] [ -S name ] directory1 directory2
```

DESCRIPTION

The diff utility will compare the contents of file1 and file2 and write to standard output a list of changes necessary to convert file1 into file2. This list should be minimal. No output will be produced if the files are identical.

The normal output contains lines of these forms:

```
n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4
```

where n1 and n2 represent lines file1 and n3 and n4 represent lines in file2 These lines resemble ed(1) commands to convert file1 to file2. By exchanging a for d and reading backward, file2 can be converted to 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

-b Ignores trailing blanks (spaces and tabs) and treats other strings of blanks as equivalent.

-i Ignores the case of letters; for example, `A' will compare equal to `a'.

-t Expands TAB characters in output lines. Normal or -c output adds character(s) to the front of each line that may adversely affect the indentation of the original source lines and make the output lines difficult to interpret. This option will preserve the original source's indentation.

-w Ignores all blanks (SPACE and TAB characters) and treats all other strings of blanks as equivalent; for example, `if ( a == b )' will compare equal to `if(a==b)'.

The following options are mutually exclusive:

-c Produces a listing of differences with three lines of context. With this option output format is modified slightly: output begins with identification of the files involved and their creation dates, then each change is separated by a line with a dozen *'s. The lines removed from file1 are marked with '-'; those added to file2 are marked '+'. Lines that are changed from one file to the other are marked in both files with '!'.

-C number Produces a listing of differences identical to that produced by -c with number lines of context.

-e Produces a script of only 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.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Except in rare circumstances, diff finds a smallest sufficient set of file differences.

-f Produces a similar script, not useful with ed, 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. Options -c, -e, -f, and -n are unavailable with -h. diff does not descend into directories with this option.

-n Produces a script similar to -e, but in the opposite order and with a count of changed lines on each insert or delete command.

-D string Creates a merged version of file1 and file2 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.

The following options are used for comparing directories:

-l Produce output in long format. Before the diff, each text file is piped through pr(1) to paginate it. Other differences are remembered and summarized after all text file differences are reported.

-r Applies diff recursively to common subdirectories encountered.

-s Reports files that are the identical; these would not otherwise be mentioned.

-S name Starts a directory diff in the middle, beginning with the file name.

OPERANDS

The following operands are supported:

file1
file2 A path name of a file or directory to be compared. If either file1 or file2 is -, the standard input will be used in its place.

directory1
directory2 A path name of a directory to be compared.

If only one of file1 and file2 is a directory, diff will be

applied to the non-directory file and the file contained in the directory file with a filename that is the same as the last component of the non-directory file.

USAGE

See largefile(5) for the description of the behavior of diff when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

If dir1 is a directory containing a directory named x, dir2 is a directory containing a directory named x, dir1/x and dir2/x both contain files named date.out, and dir2/x contains a file named y, the command:

```
example% diff -r dir1 dir2
```

could produce output similar to:

```
Common subdirectories: dir1/x and dir2/x
Only in dir2/x: y
diff -r dir1/x/date.out dir2/x/date.out
1c1
< Mon Jul  2 13:12:16 PDT 1990
---
> Tue Jun 19 21:41:39 PDT 1990
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of diff: LC_CTYPE, LC_MESSAGES, LC_TIME, and NLSPATH.

TZ Determine the locale for affecting the timezone used for calculating file timestamps written with the -C and -c options.

EXIT STATUS

The following exit values are returned:

0 No differences were found.

1 Differences were found.

>1 An error occurred.

FILES

/tmp/d????? temporary file used for comparison

/usr/lib/diffh executable file for -h option

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWesu
CSI	Enabled

SEE ALSO

bdiff(1), cmp(1), comm(1), dircmp(1), ed(1), pr(1), sdiff(1), attributes(5), environ(5), largefile(5)

NOTES

Editing scripts produced under the -e or -f options are naive about creating lines consisting of a single period (.).

Missing NEWLINE at end of file indicates that the last line of the file in question did not have a NEWLINE. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

env

env - set environment for command invocation

SYNOPSIS

```
/usr/bin/env [-i | -] [name=value] ...  [utility [ arg ...]]

/usr/xpg4/bin/env [-i | -] [name=value] ...
     [utility [ arg ...]]
```

DESCRIPTION

The env utility obtains the current environment, modifies it according to its arguments, then invokes the utility named by the utility operand with the modified environment.

Optional arguments are passed to utility. If no utility operand is specified, the resulting environment is written to the standard output, with one name=value pair per line.

/usr/bin/env

If env executes commands with arguments, it uses the default shell /usr/bin/sh (see sh(1)).

/usr/xpg4/bin/env

If env executes commands with arguments, it uses /usr/xpg4/bin/sh, which is equivalent to /usr/bin/ksh (see ksh(1)).

OPTIONS

The following options are supported:

-i | - Ignore the environment that would otherwise be inherited from the current shell. Restricts the environment for utility to that specified by the arguments.

OPERANDS

The following operands are supported:

name=value Arguments of the form name=value modify the execution environment, and are placed into the inherited environment before utility is invoked.

utility The name of the utility to be invoked. If utility names any of the special shell built-in utilities, the results are undefined.

arg A string to pass as an argument for the invoked utility.

EXAMPLES

The following utility:

```
example% env -i PATH=/mybin mygrep xyz myfile
```

invokes the utility mygrep with a new PATH value as the only entry in its environment. In this case, PATH is used to locate mygrep, which then must reside in /mybin.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of env: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

If utility is invoked, the exit status of env is the exit status of utility; otherwise, the env utility is with one of the following values:

0 Successful completion.

1-125 An error occurred.

126 utility was found but could not be invoked.

127 utility could not be found.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/env

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	enabled

|_____|_____|

/usr/xpg4/bin/env

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	enabled

SEE ALSO

ksh(1), sh(1), exec(2), profile(4), attributes(5), environ(5), xpg4(5)

expr

expr - evaluate arguments as an expression

SYNOPSIS

/usr/bin/expr argument...

/usr/xpg4/bin/expr argument...

DESCRIPTION

The expr utility evaluates the expression and writes the result to standard output. The character 0 is written to indicate a zero value and nothing is written to indicate a null string.

OPERANDS

The argument operand is evaluated as an expression. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped (see sh(1)). Strings containing blanks or other special characters should be quoted. The length of the expression is limited to LINE_MAX (2048 characters).

The operators and keywords are listed below. The list is in order of increasing precedence, with equal precedence operators grouped within {} symbols. All of the operators are left-associative.

expr \| expr
: Returns the first expr if it is neither NULL or 0, otherwise returns the second expr.

expr \& expr
: Returns the first expr if neither expr is NULL or 0, otherwise returns 0.

expr { =, \>, \>=, \<, \<=, != } expr
: Returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a string comparison using the locale-specific coalition sequence. The result of each comparison will be 1 if the specified relationship is TRUE, 0 if the relationship is FALSE.

expr { +, - } expr

Addition or subtraction of integer-valued arguments.

expr { *, /, % } expr
Multiplication, division, or remainder of the integer-valued arguments.

expr : expr The matching operator : (colon) compares the first argument with the second argument, which must be an internationalized basic regular expression (BRE); see regex(5) and NOTES. Normally, the /usr/bin/expr matching operator returns the number of bytes matched and the /usr/xpg4/bin/expr matching operator returns the number of characters matched (0 on failure). If the second argument contains at least one BRE sub-expression [\(...\)], the matching operator returns the string corresponding to \1.

integer An argument consisting only of an (optional) unary minus followed by digits.

string A string argument that cannot be identified as an integer argument or as one of the expression operator symbols.

Compatibility Operators (x86 only)

The following operators are included for compatibility with INTERACTIVE UNIX System only and are not intended to be used by non-INTERACTIVE UNIX System scripts:

index string character-list
Report the first position in which any one of the bytes in character-list matches a byte in string.

length string Return the length (that is, the number of bytes) of string.

substr string integer-1 integer-2
Extract the substring of string starting at position integer-1 and of length integer-2 bytes. If integer-1 has a value greater than the number of bytes in string, expr returns a null string. If you try to extract more bytes than there are in string, expr returns

all the remaining bytes from string. Results are unspecified if either integer-1 or integer-2 is a negative value.

EXAMPLES

Add 1 to the shell variable a:

```
example$ a=`expr $a + 1`
```

The following example emulates basename(1) - it returns the last segment of the path name $a. For $a equal to either /usr/abc/file or just file, the example returns file. (Watch out for / alone as an argument: expr takes it as the division operator; see NOTES below.)

```
example$ expr $a : '.*/\(.*\)' \| $a
```

Here is a better version of the previous example. The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

```
example$ expr //$a : '.*/\(.*\)'
```

/usr/bin/expr

Return the number of bytes in $VAR:

```
example$ expr "$VAR" : '.*'
```

/usr/xpg4/bin/expr

Return the number of characters in $VAR:

```
example$ expr "$VAR" : '.*'
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of expr: LC_COLLATE, LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

As a side effect of expression evaluation, expr returns the following exit values:

0	if the expression is neither NULL nor 0
1	if the expression is either NULL or 0

2 for invalid expressions.

>2 an error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	enabled

SEE ALSO

basename(1), ed(1), sh(1), Intro(3), attributes(5), environ(5), regex(5), xpg4(5)

DIAGNOSTICS

syntax error	Operator and operand errors.
non-numeric argument	Arithmetic is attempted on such a string.

NOTES

After argument processing by the shell, expr cannot tell the difference between an operator and an operand except by the value. If $a is an =, the command:

```
example$ expr $a = '='
```

looks like:

```
example$ expr = = =
```

as the arguments are passed to expr (and they are all taken as the = operator). The following works:

```
example$ expr X$a = X=
```

Regular Expressions

Unlike some previous versions, expr uses Internationalized Basic Regular Expressions for all system-provided locales. Internationalized Regular Expressions are explained on the regex(5) manual page.

find

find - find files

SYNOPSIS

find path... expression

DESCRIPTION

The find utility recursively descends the directory hierarchy for each path seeking files that match a Boolean expression written in the primaries given below.

find will be able to descend to arbitrary depths in a file hierarchy and will not fail due to path length limitations (unless a path operand specified by the application exceeds PATH_MAX requirements).

OPERANDS

The following operands are supported:

path — A path name of a starting point in the directory hierarchy.

expression — The first argument that starts with a -, or is a ! or a (, and all subsequent arguments will be interpreted as an expression made up of the following primaries and operators. In the descriptions, wherever n is used as a primary argument, it will be interpreted as a decimal integer optionally preceded by a plus (+) or minus (-) sign, as follows:

+n	more than n
n	exactly n
-n	less than n

Expressions

Valid expressions are:

-atime n — True if the file was accessed n days ago. The access time of directories in path is changed by find itself.

-cpio device — Always true; write the current file on device

in cpio format (5120-byte records).

-ctime n — True if the file's status was changed n days ago.

-depth — Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when find is used with cpio(1) to transfer files that are contained in directories without write permission.

-exec command — True if the executed command returns a zero value as exit status. The end of command must be punctuated by an escaped semicolon. A command argument {} is replaced by the current path name.

-follow — Always true; causes symbolic links to be followed. When following symbolic links, find keeps track of the directories visited so that it can detect infinite loops; for example, such a loop would occur if a symbolic link pointed to an ancestor. This expression should not be used with the -type l expression.

-fstype type — True if the filesystem to which the file belongs is of type type.

-group gname — True if the file belongs to the group gname. If gname is numeric and does not appear in the /etc/group file, it is taken as a group ID.

-inum n — True if the file has inode number n.

-links n — True if the file has n links.

-local — True if the file system type is not a remote file system type as defined in the /etc/dfs/fstypes file. nfs is used as the default remote filesystem type if the /etc/dfs/fstypes file is not present.

-ls — Always true; prints current path name

together with its associated statistics. These include (respectively):

+ inode number
+ size in kilobytes (1024 bytes)
+ protection mode
+ number of hard links
+ user
+ group
+ size in bytes
+ modification time.

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 `->'. The format is identical to that of ls -gilds (see ls(1)).

Note: Formatting is done internally, without executing the ls program.

-mount Always true; restricts the search to the file system containing the directory specified. Does not list mount points to other file systems.

-mtime n True if the file's data was modified n days ago.

-name pattern True if pattern matches the current file name. Normal shell file name generation characters (see sh(1)) may be used. A backslash (\) is used as an escape character within the pattern. The pattern should be escaped or quoted when find is invoked from the shell.

-ncpio device Always true; write the current file on device in cpio -c format (5120 byte records).

-newer file True if the current file has been modified more recently than the argument file.

-nogroup True if the file belongs to a group not in

the /etc/group file.

-nouser True if the file belongs to a user not in the /etc/passwd file.

-ok command Like -exec except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing y.

-perm [-]mode The mode argument is used to represent file mode bits. It will be identical in format to the <symbolicmode> operand described in chmod(1), and will be interpreted as follows. To start, a template will be assumed with all file mode bits cleared. An op symbol of:

+ will set the appropriate mode bits in the template;
- will clear the appropriate bits;

= will set the appropriate mode bits, without regard to the contents of process' file mode creation mask.

The op symbol of - cannot be the first character of mode; this avoids ambiguity with the optional leading hyphen. Since the initial mode is all bits off, there are not any symbolic modes that need to use - as the first character.

If the hyphen is omitted, the primary will evaluate as true when the file permission bits exactly match the value of the resulting template.

Otherwise, if mode is prefixed by a hyphen, the primary will evaluate as true if at least all the bits in the resulting template are set in the file permission bits.

-perm [-]onum True if the file permission flags exactly match the octal number onum (see chmod(1)). If onum is prefixed by a minus sign (-), only the bits that are set in onum are compared with the file permission flags, and the

expression evaluates true if they match.

-print Always true; causes the current path name to be printed.

-prune Always yields true. Do not examine any directories or files in the directory structure below the pattern just matched. See the examples, below.

-size n[c] True if the file is n blocks long (512 bytes per block). If n is followed by a c, the size is in bytes.

-type c True if the type of the file is c, where c is b, c, d, l, p, s, or f for block special file, character special file, directory, symbolic link, fifo (named pipe), socket, or plain file, respectively.

-user uname True if the file belongs to the user uname. If uname is numeric and does not appear as a login name in the /etc/passwd file, it is taken as a user ID.

-xdev Same as the -mount primary.

Complex Expressions

The primaries may be combined using the following operators (in order of decreasing precedence):

1) (expression) True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

2) ! expression The negation of a primary (! is the unary not operator).

3) expression [-a] expression
Concatenation of primaries (the and operation is implied by the juxtaposition of two primaries).

4) expression -o expression
Alternation of primaries (-o

is the or operator).

Note: When you use find in conjunction with cpio, if you use the -L option with cpio then you must use the -follow expression with find and vice versa. Otherwise there will be undesirable results.

If no expression is present, -print will be used as the expression. Otherwise, if the given expression does not contain any of the primaries -exec, -ok or -print, the given expression will be effectively replaced by:

```
( given_expression ) -print
```

The -user, -group, and -newer primaries each will evaluate their respective arguments only once.

USAGE

See largefile(5) for the description of the behavior of find when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

The following commands are equivalent:

```
example% find .
example% find . -print
```

They both write out the entire directory hierarchy from the current directory.

Remove all files in your home directory named a.out or *.o that have not been accessed for a week:

```
example% find $HOME \( -name a.out -o -name '*.o' \) \
     -atime +7 -exec rm {} \;
```

Recursively print all file names in the current directory and below, but skipping SCCS directories:

```
example% find . -name SCCS -prune -o -print
```

Recursively print all file names in the current directory and below, skipping the contents of SCCS directories, but printing out the SCCS directory name:

```
example% find . -print -name SCCS -prune
```

The following command is basically equivalent to the -nt extension to test(1):

```
example$ if [ -n "$(find file1 -prune -newer file2)" ]; then
        printf %s\\n "file1 is newer than file2"
```

The descriptions of -atime, -ctime, and -mtime use the terminology n ``24-hour periods''. For example, a file accessed at 23:59 will be selected by:

```
example% find . -atime -1 -print
```

at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight boundary between days has no effect on the 24-hour calculation.

Recursively print all file names whose permission mode exactly matches read, write, and execute access for user, and read and execute access for group and other.

```
example% find . -perm u=rwx,g=rx,o=rx
```

The above could alternatively be specified as follows:

```
example% find . -perm a=rwx,g-w,o-w
```

Recursively print all file names whose permission includes, but is not limited to, write access for other.

```
example% find . -perm -o+w
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of find: LC_COLLATE, LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0	All path operands were traversed successfully.
>0	An error occurred.

FILES

/etc/passwd	password file
/etc/group	group file
/etc/dfs/fstypes	file that registers distributed file system packages

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	enabled

SEE ALSO

chmod(1), cpio(1), ls(1), sh(1), test(1), stat(2), umask(2), attributes(5), environ(5), largefile(5)

WARNINGS

The following options are obsolete and will not be supported in future releases:

-cpio device Always true; write the current file on device in cpio format (5120-byte records).

-ncpio device Always true; write the current file on device in cpio -c format (5120 byte records).

NOTES

When using find to determine files modified within a range of time, one must use the ?time argument before the -print argument otherwise find will give all files.

grep

grep - search a file for a pattern

SYNOPSIS

```
/usr/bin/grep [ -bchilnsvw ] limited-regular-expression
     [ filename...  ]

/usr/xpg4/bin/grep [ -E | -F ] [ -c | -l | -q ]
     [ -bhinsvwx ] -e pattern_list
     [ -f pattern_file ] ...  [ file...  ]
/usr/xpg4/bin/grep [ -E | -F ] [ -c | -l | -q ]
     [ -bhinsvwx ] [ -e pattern_list
     -f pattern_file [ file...  ]
/usr/xpg4/bin/grep [ -E | -F ] [ -c | -l | -q ]
     [ -bhinsvwx ] pattern [ file...  ]
```

DESCRIPTION

The grep utility searches files for a pattern and prints all lines that contain that pattern. It uses a compact non-deterministic algorithm.

Be careful using the characters $, *, [, ^, |, (,), and \ in the pattern_list because they are also meaningful to the shell. It is safest to enclose the entire pattern_list in single quotes '...'.

If no files are specified, grep assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

/usr/bin/grep

The /usr/bin/grep utility uses limited regular expressions like those described on the regexp(5) manual page to match the patterns.

/usr/xpg4/bin/grep

The options -E and -F affect the way /usr/xpg4/bin/grep interprets pattern_list. If -E is specified, /usr/xpg4/bin/grep interprets pattern_list as a full regular expression (see -E for description). If -F is specified, grep interprets pattern_list as a fixed string. If neither are specified, grep interprets pattern_list as a basic regu-

lar expression as described on regex(5) manual page.

OPTIONS

The following options are supported for both /usr/bin/grep and /usr/xpg4/bin/grep:

-b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

-c Print only a count of the lines that contain the pattern.

-h Prevents the name of the file containing the matching line from being appended to that line. Used when searching multiple files.

-i Ignore upper/lower case distinction during comparisons.

-l Print only the names of files with matching lines, separated by NEWLINE characters. Does not repeat the names of files when the pattern is found more than once.

-n Precede each line by its line number in the file (first line is 1).

-s Suppress error messages about nonexistent or unreadable files.

-v Print all lines except those that contain the pattern.

-w Search for the expression as a word as if surrounded by \< and \>.

/usr/xpg4/bin/grep

The following options are supported for /usr/xpg4/bin/grep only:

-e pattern_list

Specify one or more patterns to be used during the search for input. Patterns in pattern_list must be separated by a NEWLINE

character. A null pattern can be specified by two adjacent newline characters in pattern_list. Unless the -E or -F option is also specified, each pattern will be treated as a basic regular expression. Multiple -e and -f options are accepted by grep. All of the specified patterns are used when matching lines, but the order of evaluation is unspecified.

-E Match using full regular expressions. Treat each pattern specified as a full regular expression. If any entire full regular expression pattern matches an input line, the line will be matched. A null full regular expression matches every line.

Each pattern will be interpreted as a full regular expression as described on the regex(5) manual page, except for \(and \), and including:

1. A full regular expression followed by + that matches one or more occurrences of the full regular expression.

2. A full regular expression followed by ? that matches 0 or 1 occurrences of the full regular expression.

3. Full regular expressions separated by | or by a new-line that match strings that are matched by any of the expressions.

4. A full regular expression that may be enclosed in parentheses () for grouping.

The order of precedence of operators is [], then *?+, then concatenation, then | and new-line.

-f pattern_file

Read one or more patterns from the file named by the path name pattern_file. Patterns in pattern_file are terminated by a NEWLINE character. A null pattern can be specified by an empty line in pattern_file. Unless the -E or -F option is also specified, each pat-

tern will be treated as a basic regular expression.

-F Match using fixed strings. Treat each pattern specified as a string instead of a regular expression. If an input line contains any of the patterns as a contiguous sequence of bytes, the line will be matched. A null string matches every line. See fgrep(1) for more information.

-q Quiet. Do not write anything to the standard output, regardless of matching lines. Exit with zero status if an input line is selected.

-x Consider only input lines that use all characters in the line to match an entire fixed string or regular expression to be matching lines.

OPERANDS

The following operands are supported:

file A path name of a file to be searched for the patterns. If no file operands are specified, the standard input will be used.

/usr/bin/grep

pattern Specify a pattern to be used during the search for input.

/usr/xpg4/bin/grep

pattern Specify one or more patterns to be used during the search for input. This operand is treated as if it were specified as -e pattern_list.

USAGE

The -e pattern_list option has the same effect as the pattern_list operand, but is useful when pattern_list begins with the hyphen delimiter. It is also useful when it is more convenient to provide multiple patterns as separate arguments.

Multiple -e and -f options are accepted and grep will use all of the patterns it is given while matching input text

lines. (Note that the order of evaluation is not specified. If an implementation finds a null string as a pattern, it is allowed to use that pattern first, matching every line, and effectively ignore any other patterns.)

The -q option provides a means of easily determining whether or not a pattern (or string) exists in a group of files. When searching several files, it provides a performance improvement (because it can quit as soon as it finds the first match) and requires less care by the user in choosing the set of files to supply as arguments (because it will exit zero if it finds a match even if grep detected an access or read error on earlier file operands).

Large File Behavior

See largefile(5) for the description of the behavior of grep when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

To find all uses of the word "Posix" (in any case) in the file text.mm, and write with line numbers:

```
example% /usr/bin/grep -i -n posix text.mm
```

To find all empty lines in the standard input:

```
example% /usr/bin/grep ^$
```

or

```
example% /usr/bin/grep -v .
```

Both of the following commands print all lines containing strings abc or def or both:

```
example% /usr/xpg4/bin/grep -E 'abc
def'

example% /usr/xpg4/bin/grep -F 'abc
def'
```

Both of the following commands print all lines matching exactly abc or def:

```
example% /usr/xpg4/bin/grep -E '^abc$
^def$'

example% /usr/xpg4/bin/grep -F -x 'abc
def'
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of grep: LC_COLLATE, LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0	One or more matches were found.
1	No matches were found.
2	Syntax errors or inaccessible files (even if matches were found).

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/grep

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	enabled

/usr/xpg4/bin/grep

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	enabled

SEE ALSO

egrep(1), fgrep(1), sed(1), sh(1), attributes(5), environ(5), largefile(5), regex(5), regexp(5), xpg4(5)

NOTES

/usr/bin/grep

Lines are limited only by the size of the available virtual memory. If there is a line with embedded nulls, grep will only match up to the first null; if it matches, it will print the entire line.

/usr/xpg4/bin/grep
The results are unspecified if input files contain lines longer than LINE_MAX bytes or contain binary data. LINE_MAX is defined in /usr/include/limits.a.

kill

kill - terminate or signal processes

SYNOPSIS

```
/usr/bin/kill -s signal pid...
/usr/bin/kill -l [exit_status]
/usr/bin/kill [ -signal ] pid...
```

DESCRIPTION

The kill utility sends a signal to the process or processes specified by each pid operand.

For each pid operand, the kill utility will perform actions equivalent to the kill(2) function called with the following arguments:

1. The value of the pid operand will be used as the pid argument.

2. The sig argument is the value specified by the -s option, or by SIGTERM, if none of these options is specified.

The signaled process must belong to the current user unless the user is the super-user.

See NOTES for descriptions of the shell built-in versions of kill.

OPTIONS

The following options are supported:

-l (The letter ell.) Write all values of signal supported by the implementation, if no operand is given. If an exit_status operand is given and it is a value of the ? shell special parameter and wait corresponding to a process that was terminated by a signal, the signal corresponding to the signal that terminated the process will be written. If an exit_status operand is given and it is the unsigned decimal integer value of a signal

number, the signal corresponding to that signal will be written. Otherwise, the results are unspecified.

-s signal Specify the signal to send, using one of the symbolic names defined in the <signal.h> description. Values of signal will be recognized in a case-independent fashion, without the SIG prefix. In addition, the symbolic name 0 will be recognized, representing the signal value zero. The corresponding signal will be sent instead of SIGTERM.

OPERANDS

The following operands are supported:

pid One of the following:

1. A decimal integer specifying a process or process group to be signaled. The process or processes selected by positive, negative and zero values of the pid operand will be as described for the kill function. If process number 0 is specified, all processes in the process group are signaled. If the first pid operand is negative, it should be preceded by -- to keep it from being interpreted as an option.

2. A job control job ID that identifies a background process group to be signaled. The job control job ID notation is applicable only for invocations of kill in the current shell execution environment.

Note the job control job ID type of pid is available only on systems supporting the job control option.

exit_status A decimal integer specifying a signal number or the exit status of a process terminated by a signal.

USAGE

Process numbers can be found by using ps(1).

The job control job ID notation is not required to work as expected when kill is operating in its own utility execution environment. In either of the following examples:

```
nohup kill %1 &
system( "kill %1");
```

kill operates in a different environment and will not share the shell's understanding of job numbers.

OUTPUT

When the -l option is not specified, the standard output will not be used.

When the -l option is specified, the symbolic name of each signal will be written in the following format:

```
"%s%c", <signal>, <separator>
```

where the <signal> is in upper-case, without the SIG prefix, and the <separator> will be either a newline character or a space character. For the last signal written, <separator> will be a newline character.

When both the -l option and exit_status operand are specified, the symbolic name of the corresponding signal will be written in the following format:

```
"%s\n", <signal>
```

EXAMPLES

Any of the commands:

```
kill -9 100 -165

kill -s kill 100 -165

kill -s KILL 100 -165
```

sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose process group ID is 165, assuming the sending process has permission to send that signal to the specified processes, and that they exist.

To avoid an ambiguity of an initial negative number argument specifying either a signal number or a process group, the former will always be the case. Therefore, to send the default signal to a process group (for example, 123), an

application should use a command similar to one of the following:

```
kill -TERM -123

kill -- -123
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of kill: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 At least one matching process was found for each pid operand, and the specified signal was successfully processed for at least one matching process.

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	enabled

SEE ALSO

csh(1), jobs(1), ksh(1), ps(1), sh(1), shell_builtins(1), wait(1), kill(2), signal(3C), attributes(5), environ(5), signal(5)

NOTES

sh

The Bourne shell, sh, has a built-in version of kill to provide the functionality of the kill command for processes identified with a jobid. The sh syntax is:

```
kill [ -sig ] [ pid ] [ %job ] ...
kill -l
```

csh

The C-shell, csh, also has a built-in kill command, whose syntax is:

```
kill [ -sig ] [ pid ] [ %job ] ...
kill -l
```

The csh kill built-in sends the TERM (terminate) signal, by default, or the signal specified, to the specified process ID, the job indicated, or the current job. Signals are either given by number or by name. There is no default. Typing kill does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process is sent a CONT (continue) signal as well.

-l List the signal names that can be sent.

ksh

The ksh kill's syntax is:

```
kill [ -sig ] [ pid ] [ %job ] ...
kill -l
```

The ksh kill sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in signal(5) stripped of the prefix "SIG"). If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal if it is stopped. The argument job can be the process id of a process that is not a member of one of the active jobs. In the second form, kill -l, the signal numbers and names are listed.

ksh

ksh, rksh - KornShell, a standard/restricted command and programming language

SYNOPSIS

```
/usr/bin/ksh [ +-abCefhikmnoprstuvx ] [ +-o option ] ...
     [ -c string ] [ arg...  ]

/usr/xpg4/bin/sh [ +-abCefhikmnoprstuvx ] [ +-o option ] ...
     [ -c string ] [ arg...  ]

/usr/bin/rksh [ +-abCefhikmnoprstuvx ] [ +-o option ] ...
     [ -c string ] [ arg...  ]
```

DESCRIPTION

/usr/xpg4/bin/sh is identical to /usr/bin/ksh, a command and programming language that executes commands read from a terminal or a file. rksh is a restricted version of the command interpreter ksh; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See Invocation below for the meaning of arguments to the shell.

Definitions

A metacharacter is one of the following characters:

```
;   &   (   )   |   <   >     NEWLINE   SPACE   TAB
```

A blank is a TAB or a SPACE. An identifier is a sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for functions and variables. A word is a sequence of characters separated by one or more non-quoted metacharacters.

A command is a sequence of characters in the syntax of the shell language. The shell reads each command and carries out the desired action either directly or by invoking separate utilities. A special-command is a command that is carried out by the shell without creating a separate process. Except for documented side effects, most special commands can be implemented as separate utilities.

Commands

A simple-command is a sequence of blank-separated words which may be preceded by a variable assignment list. (See Environment below.) The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see exec(2)). The value of a simple-command is its exit status if it terminates normally, or (octal) 200+status if it terminates abnormally (see signal(3C) for a list of status values).

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

A list is a sequence of one or more pipelines separated by ;, &, &&, or ||, and optionally terminated by ;, &, or |&. Of these five symbols, ;, &, and |& have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (that is, the shell does not wait for that pipeline to finish). The symbol |& causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell.

The standard input and output of the spawned command can be written to and read from by the parent shell using the -p option of the special commands read and print described in Special Commands. The symbol && (||) causes the list following it to be executed only if the preceding pipeline returns 0 (or a non-zero) value. An arbitrary number of new-lines may appear in a list, instead of a semicolon, to delimit a command.

A command is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for identifier [in word ...] ; do list ; done
Each time a for command is executed, identifier is set to the next word taken from the in word list. If in

word ... is omitted, then the for command executes the do list once for each positional parameter that is set (see Parameter Substitution below). Execution ends when there are no more words in the list.

select identifier [in word ...] ; do list ; done
A select command prints to standard error (file descriptor 2), the set of words, each preceded by a number. If in word ... is omitted, then the positional parameters are used instead (see Parameter Substitution below). The PS3 prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed words, then the value of the variable identifier is set to the word corresponding to this number. If this line is empty the selection list is printed again. Otherwise the value of the variable identifier is set to NULL. (See Blank Interpretation about NULL). The contents of the line read from standard input is saved in the shell variable REPLY. The list is executed for each selection until a break or EOF is encountered. If the REPLY variable is set to NULL by the execution of list, then the selection list is printed before displaying the PS3 prompt for the next selection.

case word in [pattern [| pattern]) list ;;] ... esac
A case command executes the list associated with the first pattern that matches word. The form of the patterns is the same as that used for file-name generation (see File Name Generation below).

if list ; then list ;
[elif list ; then list ; ...] [else list ;] fi
The list following if is executed and, if it returns an exit status of 0, the list following the first then is executed. Otherwise, the list following elif is executed and, if its value is 0, the list following the next then is executed. Failing that, the else list is executed. If no else list or then list is executed, then the if command returns 0 exit status.

while list ; do list ; done
until list ; do list ; done
A while command repeatedly executes the while list and, if the exit status of the last command in the list is 0, executes the do list; otherwise the loop terminates. If no commands in the do list are executed, then the

while command returns 0 exit status; until may be used in place of while to negate the loop termination test.

(list)
Execute list in a separate environment. Note, that if two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below.

{list}
list is simply executed. Note that unlike the meta-characters (and), { and } are reserved words and must occur at the beginning of a line or after a ; in order to be recognized.

[[expression]]
Evaluates expression and returns 0 exit status when expression is true. See Conditional Expressions below, for a description of expression.

function identifier { list ;}
identifier() { list ;}
Define a function which is referenced by identifier. The body of the function is the list of commands between { and }. (See Functions below).

time pipeline
The pipeline is executed and the elapsed time as well as the user and system time are printed to standard error.

The following reserved words are only recognized as the first word of a command and when not quoted:

! if then else elif fi case esac for while until do done { } function select time [[ ]]

Comments

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Aliasing

The first word of each command is replaced by the text of an alias if an alias for this word has been defined. An alias name consists of any number of characters excluding meta-characters, quoting characters, file expansion characters, parameter and command substitution characters, and =. The replacement string can contain any valid shell script

including the metacharacters listed above. The first word of each command in the replaced text, other than any that are in the process of being replaced, will be tested for aliases. If the last character of the alias value is a blank then the word following the alias will also be checked for alias substitution. Aliases can be used to redefine special builtin commands but cannot be used to redefine the reserved words listed above. Aliases can be created, listed, and exported with the alias command and can be removed with the unalias command. Exported aliases remain in effect for scripts invoked by name, but must be reinitialized for separate invocations of the shell (see Invocation below). To prevent infinite loops in recursive aliasing, if the shell is not currently processing an alias of the same name, the word will be replaced by the value of the alias; otherwise, it will not be replaced.

Aliasing is performed when scripts are read, not while they are executed. Therefore, for an alias to take effect the alias definition command has to be executed before the command which references the alias is read.

Aliases are frequently used as a short hand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full pathname of the corresponding command. These aliases are called tracked aliases. The value of a tracked alias is defined the first time the corresponding command is looked up and becomes undefined each time the PATH variable is reset. These aliases remain tracked so that the next subsequent reference will redefine the value. Several tracked aliases are compiled into the shell. The -h option of the set command makes each referenced command name into a tracked alias.

The following exported aliases are compiled into (and built-in to) the shell but can be unset or redefined:

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
```

An example concerning trailing blank characters and reserved words follows. If the user types:

```
$ alias foo="/bin/ls "
$ alias while="/"
```

The effect of executing:

```
$ while true
> do
> echo "Hello, World"
> done
```

is a never-ending sequence of Hello, World strings to the screen. However, if the user types:

```
$ foo while
```

the result will be an ls listing of /. Since the alias substitution for foo ends in a space character, the next word is checked for alias substitution. The next word, while, has also been aliased, so it is substituted as well. Since it is not in the proper position as a command word, it is not recognized as a reserved word.

If the user types:

```
$ foo; while
```

while retains its normal reserved-word properties.

Tilde Substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~. If it does, then the word up to a / is checked to see if it matches a user name. If a match is found, the ~ and the matched login name are replaced by the login directory of the matched user. This is called a tilde substitution. If no match is found, the original text is left unchanged. A ~ by itself, or in front of a /, is replaced by $HOME. A ~ followed by a + or - is replaced by $PWD and $OLDPWD respectively.

In addition, tilde substitution is attempted when the value of a variable assignment begins with a ~.

Tilde Expansion

A tilde-prefix consists of an unquoted tilde character at the beginning of a word, followed by all of the characters

preceding the first unquoted slash in the word, or all the characters in the word if there is no slash. In an assignment, multiple tilde-prefixes can be used: at the beginning of the word (that is, following the equal sign of the assignment), following any unquoted colon or both. A tilde-prefix in an assignment is terminated by the first unquoted colon or slash. If none of the characters in the tilde-prefix are quoted, the characters in the tilde-prefix following the tilde are treated as a possible login name from the user database.

A portable login name cannot contain characters outside the set given in the description of the LOGNAME ENVIRONMENT variable. If the login name is null (that is, the tilde-prefix contains only the tilde), the tilde-prefix will be replaced by the value of the variable HOME. If HOME is unset, the results are unspecified. Otherwise, the tilde-prefix will be replaced by a pathname of the home directory associated with the login name obtained using the getpwnam function. If the system does not recognize the login name, the results are undefined.

Tilde expansion generally occurs only at the beginning of words, but an exception based on historical practice has been included:

```
PATH=/posix/bin:~dgk/bin
```

is eligible for tilde expansion because tilde follows a colon and none of the relevant characters is quoted. Consideration was given to prohibiting this behavior because any of the following are reasonable substitutes:

```
PATH=$(printf %s ~karels/bin : ~bostic/bin)
for Dir in ~maart/bin ~srb/bin ...
do
        PATH=${PATH:+$PATH:}$Dir
done
```

With the first command, explicit colons are used for each directory. In all cases, the shell performs tilde expansion on each directory because all are separate words to the shell.

Note that expressions in operands such as:

```
make -k mumble LIBDIR=~chet/lib
```

do not qualify as shell variable assignments and tilde expansion is not performed (unless the command does so itself, which make does not).

The special sequence $~ has been designated for future implementations to evaluate as a means of forcing tilde expansion in any word.

Because of the requirement that the word not be quoted, the following are not equivalent; only the last will cause tilde expansion:

```
\~hlj/   ~h\lj/   ~"hlj"/   ~hlj\/   ~hlj/
```

The results of giving tilde with an unknown login name are undefined because the KornShell ~+ and ~- constructs make use of this condition, but, in general it is an error to give an incorrect login name with tilde. The results of having HOME unset are unspecified because some historical shells treat this as an error.

Command Substitution

The standard output from a command enclosed in parenthesis preceded by a dollar sign ($(command)) or a pair of grave accents (``) may be used as part or all of a word; trailing new-lines are removed. In the second (archaic) form, the string between the quotes is processed for special quoting characters before the command is executed. (See Quoting below.) The command substitution $(cat file) can be replaced by the equivalent but faster (<file). Command substitution of most special commands that do not perform input/output redirection are carried out without creating a separate process.

Command substitution allows the output of a command to be substituted in place of the command name itself. Command substitution occurs when the command is enclosed as follows:

```
$(command)
```

or (backquoted version):

```
`command`
```

The shell will expand the command substitution by executing command in a subshell environment and replacing the command substitution (the text of command plus the enclosing $( ) or backquotes) with the standard output of the command, remov-

ing sequences of one or more newline characters at the end of the substitution. Embedded newline characters before the end of the output will not be removed; however, they may be treated as field delimiters and eliminated during field splitting, depending on the value of IFS and quoting that is in effect.

Within the backquoted style of command substitution, backslash shall retain its literal meaning, except when followed by:

```
$     `     \
```

(dollar-sign, backquote, backslash). The search for the matching backquote is satisfied by the first backquote found without a preceding backslash; during this search, if a non-escaped backquote is encountered within a shell comment, a here-document, an embedded command substitution of the $(command) form, or a quoted string, undefined results occur. A single- or double-quoted string that begins, but does not end, within the `...` sequence produces undefined results.

With the $(command) form, all characters following the open parenthesis to the matching closing parenthesis constitute the command. Any valid shell script can be used for command, except:

+ A script consisting solely of redirections produces unspecified results.

+ See the restriction on single subshells described below.

The results of command substitution will not be field splitting and pathname expansion processed for further tilde expansion, parameter expansion, command substitution or arithmetic expansion. If a command substitution occurs inside double-quotes, it will not be performed on the results of the substitution.

Command substitution can be nested. To specify nesting within the backquoted version, the application must precede the inner backquotes with backslashes; for example:

```
`\`command\``
```

The $( ) form of command substitution solves a problem of

inconsistent behavior when using backquotes. For example:

Command	Output
`echo '\$x'`	`\$x`
`echo `echo '\$x'``	`$x`
`echo $(echo '\$x')`	`\$x`

Additionally, the backquoted syntax has historical restrictions on the contents of the embedded command. While the new $( ) form can process any kind of valid embedded script, the backquoted form cannot handle some valid scripts that include backquotes. For example, these otherwise valid embedded scripts do not work in the left column, but do work on the right:

```
echo `                          echo $(
cat <<eeof                      cat <<eeof
a here-doc with `               a here-doc with )
eof                             eof
`                               )

echo `                          echo $(
echo abc # a comment with `     echo abc # a comment with )
`                               )

echo `                          echo $(
echo '`'                        echo ')'
`                               )
```

Because of these inconsistent behaviors, the backquoted variety of command substitution is not recommended for new applications that nest command substitutions or attempt to embed complex scripts.

If the command substitution consists of a single subshell, such as:

```
$( (command) )
```

a portable application must separate the $(and (into two tokens (that is, separate them with white space). This is required to avoid any ambiguities with arithmetic expansion.

Arithmetic Expansion

An arithmetic expression enclosed in double parentheses preceded by a dollar sign ($((arithmetic-expression))) is

replaced by the value of the arithmetic expression within the double parenthesis. Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and substituting its value. The format for arithmetic expansion is as follows:

```
$((expression))
```

The expression is treated as if it were in double-quotes, except that a double-quote inside the expression is not treated specially. The shell will expand all tokens in the expression for parameter expansion, command substitution and quote removal.

Next, the shell will treat this as an arithmetic expression and substitute the value of the expression. The arithmetic expression will be processed according to the rules of the ISO C with the following exceptions:

+ Only integer arithmetic is required.

+ The sizeof() operator and the prefix and postfix ++ and -- operators are not required.

+ Selection, iteration and jump statements are not supported.

As an extension, the shell may recognize arithmetic expressions beyond those listed. If the expression is invalid, the expansion will fail and the shell will write a message to standard error indicating the failure.

A simple example using arithmetic expansion:

```
# repeat a command 100 times
x=100
while [ $x -gt 0 ]
do
        command
        x=$(($x-1))
done
```

Process Substitution

This feature is available in SunOS and only on versions of the UNIX operating system that support the /dev/fd directory for naming open files. Each command argument of the form <(list) or >(list) will run process list asynchronously con-

nected to some file in /dev/fd. The name of this file will become the argument to the command. If the form with > is selected then writing on this file will provide input for list. If < is used, then the file passed as an argument will contain the output of the list process. For example,

```
paste <(cut -f1 file1) <(cut -f3 file2) |
    tee >( process1) >( process2)
```

cuts fields 1 and 3 from the files file1 and file2 respectively, pastes the results together, and sends it to the processes process1 and process2, as well as putting it onto the standard output. Note that the file, which is passed as an argument to the command, is a UNIX pipe(2) so programs that expect to lseek(2) on the file will not work.

Parameter Substitution

A parameter is an identifier, one or more digits, or any of the characters *, @, #, ?, -, $, and !. A variable (a parameter denoted by an identifier) has a value and zero or more attributes. variables can be assigned values and attributes by using the typeset special command. The attributes supported by the shell are described later with the typeset special command. Exported variables pass values and attributes to the environment.

The shell supports a one-dimensional array facility. An element of an array variable is referenced by a subscript. A subscript is denoted by a [, followed by an arithmetic expression (see Arithmetic Evaluation below) followed by a]. To assign values to an array, use set -A name value .... The value of all subscripts must be in the range of 0 through 1023. Arrays need not be declared. Any reference to a variable with a valid subscript is legal and an array will be created if necessary. Referencing an array without a subscript is equivalent to referencing the element 0. If an array identifier with subscript * or @ is used, then the value for each of the elements is substituted (separated by a field separator character).

The value of a variable may be assigned by writing:

```
name=value [ name=value ] ...
```

If the integer attribute, -i, is set for name, the value is subject to arithmetic evaluation as described below.

Positional parameters, parameters denoted by a number, may

be assigned values with the set special command. Parameter $0 is set from argument zero when the shell is invoked. If parameter is one or more digits then it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces.

Parameter Expansion

The format for parameter expansion is as follows:

```
${expression}
```

where expression consists of all characters until the matching }. Any } escaped by a backslash or within a quoted string, and characters in embedded arithmetic expansions, command substitutions and variable expansions, are not examined in determining the matching }.

The simplest form for parameter expansion is:

```
${parameter}
```

The value, if any, of parameter will be substituted.

The parameter name or symbol can be enclosed in braces, which are optional except for positional parameters with more than one digit or when parameter is followed by a character that could be interpreted as part of the name. The matching closing brace will be determined by counting brace levels, skipping over enclosed quoted strings and command substitutions.

If the parameter name or symbol is not enclosed in braces, the expansion will use the longest valid name whether or not the symbol represented by that name exists. When the shell is scanning its input to determine the boundaries of a name, it is not bound by its knowledge of what names are already defined. For example, if F is a defined shell variable, the command:

```
echo $Fred
```

does not echo the value of $F followed by red; it selects the longest possible valid name, Fred, which in this case might be unset.

If a parameter expansion occurs inside double-quotes:

+ Pathname expansion will not be performed on the results

of the expansion.

\+ Field splitting will not be performed on the results of the expansion, with the exception of @.

In addition, a parameter expansion can be modified by using one of the following formats. In each case that a value of word is needed (based on the state of parameter, as described below), word will be subjected to tilde expansion, parameter expansion, command substitution and arithmetic expansion. If word is not needed, it will not be expanded. The } character that delimits the following parameter expansion modifications is determined as described previously in this section and in dquote. (For example, ${foo-bar}xyz} would result in the expansion of foo followed by the string xyz} if foo is set, else the string barxyz}).

${parameter:-word} Use Default Values. If parameter is unset or null, the expansion of word will be substituted; otherwise, the value of parameter will be substituted.

${parameter:=word} Assign Default Values. If parameter is unset or null, the expansion of word will be assigned to parameter. In all cases, the final value of parameter will be substituted. Only variables, not positional parameters or special parameters, can be assigned in this way.

${parameter:?[word]}
Indicate Error if Null or Unset. If parameter is unset or null, the expansion of word (or a message indicating it is unset if word is omitted) will be written to standard error and the shell will exit with a non-zero exit status. Otherwise, the value of parameter will be substituted. An interactive shell need not exit.

${parameter:+[word]}
Use Alternative Value. If parameter is unset or null, null will be substituted; otherwise, the expansion of word will be substituted.

In the parameter expansions shown previously, use of the colon in the format results in a test for a parameter that is unset or null; omission of the colon results in a test for a parameter that is only unset. The following table summarizes the effect of the colon:

	set and not null	set but null	unset
${parameter:-word}	substitute parameter	substitute word	substitute word
${parameter-word}	substitute parameter	substitute null	substitute word
${parameter:=word}	substitute parameter	assign word	assign word
${parameter=word}	substitute parameter	substitute parameter	assign null
${parameter:?word}	substitute parameter	error, exit	error, exit
${parameter?word}	substitute parameter	substitute null	error, exit
${parameter:+word}	substitute word	substitute null	substitute null
${parameter+word}	substitute word	substitute word	substitute null

In all cases shown with "substitute", the expression is replaced with the value shown. In all cases shown with "assign" parameter is assigned that value, which also replaces the expression.

${#parameter} String Length. The length in characters of the value of parameter. If parameter is * or @, then all the positional parameters, starting with $1, are substituted (separated by a field separator charac-

ter).

The following four varieties of parameter expansion provide for substring processing. In each case, pattern matching notation (see patmat), rather than regular expression notation, will be used to evaluate the patterns. If parameter is * or @, then all the positional parameters, starting with $1, are substituted (separated by a field separator character). Enclosing the full parameter expansion string in double-quotes will not cause the following four varieties of pattern characters to be quoted, whereas quoting characters within the braces will have this effect.

${parameter%word} Remove Smallest Suffix Pattern. The word will be expanded to produce a pattern. The parameter expansion then will result in parameter, with the smallest portion of the suffix matched by the pattern deleted.

${parameter%%word} Remove Largest Suffix Pattern. The word will be expanded to produce a pattern. The parameter expansion then will result in parameter, with the largest portion of the suffix matched by the pattern deleted.

${parameter#word} Remove Smallest Prefix Pattern. The word will be expanded to produce a pattern. The parameter expansion then will result in parameter, with the smallest portion of the prefix matched by the pattern deleted.

${parameter##word} Remove Largest Prefix Pattern. The word will be expanded to produce a pattern. The parameter expansion then will result in parameter, with the largest portion of the prefix matched by the pattern deleted.

Examples:

${parameter:-word}

In this example, ls is executed only if x is null or unset. (The $(ls) command substitution notation is explained in Command Substitution above.)

```
     ${x:-$(ls)}

${parameter:=word}

     unset X
     echo ${X:=abc}
     abc

${parameter:?word}

     unset posix
     echo ${posix:?}
     sh: posix: parameter null or not set

${parameter:+word}

     set a b c
     echo ${3:+posix}
     posix

${#parameter}

     HOME=/usr/posix
     echo ${#HOME}
     10

${parameter%word}

     x=file.c
     echo ${x%.c}.o
     file.o

${parameter%%word}
     x=posix/src/std
     echo ${x%%/*}
     posix

${parameter#word}

     x=$HOME/src/cmd
     echo ${x#$HOME}
     /src/cmd

${parameter##word}

     x=/one/two/three
```

```
echo ${x##*/}
three
```

Parameters Set by Shell

The following parameters are automatically set by the shell:

`#` The number of positional parameters in decimal.

`-` Flags supplied to the shell on invocation or by the set command.

`?` The decimal value returned by the last executed command.

`$` The process number of this shell.

`_` Initially, the value of _ is an absolute pathname of the shell or script being executed as passed in the environment. Subsequently it is assigned the last argument of the previous command. This parameter is not set for commands which are asynchronous. This parameter is also used to hold the name of the matching MAIL file when checking for mail.

`!` The process number of the last background command invoked.

`ERRNO` The value of errno as set by the most recently failed system call. This value is system dependent and is intended for debugging purposes.

`LINENO` The line number of the current line within the script or function being executed.

`OLDPWD` The previous working directory set by the cd command.

`OPTARG` The value of the last option argument processed by the getopts special command.

`OPTIND` The index of the last option argument processed by the getopts special command.

PPID The process number of the parent of the shell.

PWD The present working directory set by the cd command.

RANDOM Each time this variable is referenced, a random integer, uniformly distributed between 0 and 32767, is generated. The sequence of random numbers can be initialized by assigning a numeric value to RANDOM.

REPLY This variable is set by the select statement and by the read special command when no arguments are supplied.

SECONDS Each time this variable is referenced, the number of seconds since shell invocation is returned. If this variable is assigned a value, then the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

Variables Used by Shell

The following variables are used by the shell:

CDPATH The search path for the cd command.

COLUMNS If this variable is set, the value is used to define the width of the edit window for the shell edit modes and for printing select lists.

EDITOR If the value of this variable ends in emacs, gmacs, or vi and the VISUAL variable is not set, then the corresponding option (see the set special command below) will be turned on.

ENV This variable, when the shell is invoked, is subjected to parameter expansion by the shell and the resulting value is used as a pathname of a file containing shell commands to execute in the current environment. The file need not be executable. If the expanded value of ENV is not an absolute pathname, the results are unspecified. ENV will be ignored if the user's real and effective user IDs or

real and effective group IDs are different.

This variable can be used to set aliases and other items local to the invocation of a shell. The file referred to by ENV differs from $HOME/.profile in that .profile is typically executed at session startup, whereas the ENV file is executed at the beginning of each shell invocation. The ENV value is interpreted in a manner similar to a dot script, in that the commands are executed in the current environment and the file needs to be readable, but not executable. However, unlike dot scripts, no PATH searching is performed. This is used as a guard against Trojan Horse security breaches.

FCEDIT The default editor name for the fc command.

FPATH The search path for function definitions. By default the FPATH directories are searched after the PATH variable. If an executable file is found, then it is read and executed in the current environment. FPATH is searched before PATH when a function with the -u attribute is referenced. The preset alias autoload preset alias causes a function with the -u attribute to be created.

IFS Internal field separators, normally space, tab, and new-line that are used to separate command words which result from command or parameter substitution and for separating words with the special command read. The first character of the IFS variable is used to separate arguments for the $* substitution (See Quoting below).

HISTFILE If this variable is set when the shell is invoked, then the value is the pathname of the file that will be used to store the command history. (See Command re-entry below.)

HISTSIZE If this variable is set when the shell is invoked, then the number of previously entered commands that are accessible by this shell will be greater than or equal to this

number. The default is 128.

HOME The default argument (home directory) for the cd command.

LC_ALL This variable provides a default value for the LC_* variables.

LC_COLLATE
This variable determines the behavior of range expressions, equivalence classes and multi-byte character collating elements within pattern matching.

LC_CTYPE Determines how the shell handles characters. When LC_CTYPE is set to a valid value, the shell can display and handle text and filenames containing valid characters for that locale. If LC_CTYPE (see environ(5)) is not set in the environment, the operational behavior of the shell is determined by the value of the LANG environment variable. If LC_ALL is set, its contents are used to override both the LANG and the other LC_* variables.

LC_MESSAGES
This variable determines the language in which messages should be written.

LANG Provide a default value for the internationalization variables that are unset or null. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LINENO This variable is set by the shell to a decimal number representing the current sequential line number (numbered starting with 1) within a script or function before it executes each command. If the user unsets or resets LINENO, the variable may lose its special meaning for the life of the shell. If the shell is not currently executing a script or function, the value of LINENO is unspecified.

LINES If this variable is set, the value is used to determine the column length for printing select lists. Select lists will print vertically until about two-thirds of LINES lines are filled.

MAIL If this variable is set to the name of a mail file and the MAILPATH variable is not set, then the shell informs the user of arrival of mail in the specified file.

MAILCHECK This variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the MAILPATH or MAIL variables. The default value is 600 seconds. When the time has elapsed the shell will check before issuing the next prompt.

MAILPATH A colon (:) separated list of file names. If this variable is set, then the shell informs the user of any modifications to the specified files that have occurred within the last MAILCHECK seconds. Each file name can be followed by a ? and a message that will be printed. The message will undergo parameter substitution with the variable $_ defined as the name of the file that has changed. The default message is you have mail in $_.

NLSPATH Determine the location of message catalogues for the processing of LC_MESSAGES.

PATH The search path for commands (see Execution below). The user may not change PATH if executing under rksh (except in .profile).

PPID This variable is set by the shell to the decimal process ID of the process that invoked the shell. In a subshell, PPID will be set to the same value as that of the parent of the current shell. For example, echo $PPID and (echo $PPID) would produce the same value.

PS1 The value of this variable is expanded for

parameter substitution to define the primary prompt string which by default is ``$ ''. The character ! in the primary prompt string is replaced by the command number (see Command Re-entry below). Two successive occurrences of ! will produce a single ! when the prompt string is printed.

PS2 Secondary prompt string, by default ``> ''.

PS3 Selection prompt string used within a select loop, by default ``#? ''.

PS4 The value of this variable is expanded for parameter substitution and precedes each line of an execution trace. If omitted, the execution trace prompt is ``+ ''.

SHELL The pathname of the shell is kept in the environment. At invocation, if the basename of this variable is rsh, rksh, or krsh, then the shell becomes restricted.

TMOUT If set to a value greater than zero, the shell will terminate if a command is not entered within the prescribed number of seconds after issuing the PS1 prompt. (Note that the shell can be compiled with a maximum bound for this value which cannot be exceeded.)

VISUAL If the value of this variable ends in emacs, gmacs, or vi then the corresponding option (see Special Command set below) will be turned on.

The shell gives default values to PATH, PS1, PS2, PS3, PS4, MAILCHECK, FCEDIT, TMOUT and IFS, while HOME, SHELL ENV and MAIL are not set at all by the shell (although HOME is set by login(1)). On some systems MAIL and SHELL are also set by login.

Blank Interpretation

After parameter and command substitution, the results of substitutions are scanned for the field separator characters (those found in IFS) and split into distinct arguments where such characters are found. Explicit null arguments ("") or ('') are retained. Implicit null arguments (those

resulting from parameters that have no values) are removed.

File Name Generation

Following substitution, each command word is scanned for the characters *, ?, and [unless the -f option has been set. If one of these characters appears then the word is regarded as a pattern. The word is replaced with lexicographically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. When a pattern is used for file name generation, the character period (.) at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly. A file name beginning with a period will not be matched with a pattern with the period inside parentheses; that is

```
ls .@(r*)
```

would locate a file named .restore, but ls @(.r*) would not. In other instances of pattern matching the / and . are not treated specially.

* Matches any string, including the null string.

? Matches any single character.

[...]
Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "! ", then any character not enclosed is matched. A - can be included in the character set by putting it as the first or last character.

A pattern-list is a list of one or more patterns separated from each other with a |. Composite patterns can be formed with one or more of the following:

?(pattern-list)	Optionally matches any one of the given patterns.
*(pattern-list)	Matches zero or more occurrences of the given patterns.
+(pattern-list)	Matches one or more occurrences of the given patterns.
@(pattern-list)	Matches exactly one of the given patterns.
!(pattern-list)	Matches anything, except one of the given patterns.

Quoting

Each of the metacharacters listed above (See Definitions) has a special meaning to the shell and causes termination of a word unless quoted. A character may be quoted (that is, made to stand for itself) by preceding it with a \. The pair \NEWLINE is removed. All characters enclosed between a pair of single quote marks ('') are quoted. A single quote cannot appear within single quotes. Inside double quote marks (""), parameter and command substitution occur and \ quotes the characters \, `, ", and $. The meaning of $* and $@ is identical when not quoted or when used as a parameter assignment value or as a file name. However, when used as a command argument, $* is equivalent to ``$1d$2d...'', where d is the first character of the IFS variable, whereas $@ is equivalent to $1 $2 .... Inside grave quote marks (``), \ quotes the characters \, `, and $. If the grave quotes occur within double quotes, then \ also quotes the character ".

The special meaning of reserved words or aliases can be removed by quoting any character of the reserved word. The recognition of function names or special command names listed below cannot be altered by quoting them.

Arithmetic Evaluation

An ability to perform integer arithmetic is provided with the special command let. Evaluations are performed using long arithmetic. Constants are of the form [base#] n where base is a decimal number between two and thirty-six representing the arithmetic base and n is a number in that base. If base is omitted then base 10 is used.

An arithmetic expression uses the same syntax, precedence, and associativity of expression as the C language. All the integral operators, other than ++, --, ?:, and , are supported. Variables can be referenced by name within an arithmetic expression without using the parameter substitution syntax. When a variable is referenced, its value is evaluated as an arithmetic expression.

An internal integer representation of a variable can be specified with the -i option of the typeset special command. Arithmetic evaluation is performed on the value of each assignment to a variable with the -i attribute. If you do not specify an arithmetic base, the first assignment to the variable determines the arithmetic base. This base is used when parameter substitution occurs.

Since many of the arithmetic operators require quoting, an alternative form of the let command is provided. For any command which begins with a ((, all the characters until a matching)) are treated as a quoted expression. More precisely, ((...)) is equivalent to let "...".

Prompting

When used interactively, the shell prompts with the parameter expanded value of PS1 before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (that is, the value of PS2) is issued.

Conditional Expressions

A conditional expression is used with the [[compound command to test attributes of files and to compare strings. Word splitting and file name generation are not performed on the words between [[and]]. Each expression can be constructed from one or more of the following unary or binary expressions:

-a file	True, if file exists.
-b file	True, if file exists and is a block special file.
-c file	True, if file exists and is a character special file.
-d file	True, if file exists and is a directory.
-e file	True, if file exists.
-f file	True, if file exists and is an ordinary file.
-g file	True, if file exists and is has its setgid bit set.
-k file	True, if file exists and is has its sticky bit set.
-n string	True, if length of string is non-zero.
-o option	True, if option named option is on.
-p file	True, if file exists and is a fifo special file or a pipe.
-r file	True, if file exists and is readable by current process.
-s file	True, if file exists and has size greater than zero.
-t fildes	True, if file descriptor number fildes is open and associated with a terminal device.
-u file	True, if file exists and has its setuid

	bit set.
-w file	True, if file exists and is writable by current process.
-x file	True, if file exists and is executable by current process. If file exists and is a directory, then the current process has permission to search in the directory.
-z string	True, if length of string is zero.
-L file	True, if file exists and is a symbolic link.
-O file	True, if file exists and is owned by the effective user id of this process.
-G file	True, if file exists and its group matches the effective group id of this process.
-S file	True, if file exists and is a socket.
file1 -nt file2	True, if file1 exists and is newer than file2.
file1 -ot file2	True, if file1 exists and is older than file2.
file1 -ef file2	True, if file1 and file2 exist and refer to the same file.
string	True if the string string is not the null string.
string = pattern	True, if string matches pattern.
string != pattern	True, if string does not match pattern.
string1 = string2	True if the strings string1 and string2 are identical.
string1 ! = string2	True if the strings string1 and string2 are not identical.
string1 < string2	True, if string1 comes before string2 based on strings interpreted as appropriate to the locale setting for category LC_COLLATE.
string1 > string2	True, if string1 comes after string2 based on strings interpreted as appropriate to the locale setting for category LC_COLLATE.
exp1 -eq exp2	True, if exp1 is equal to exp2.
exp1 -ne exp2	True, if exp1 is not equal to exp2.
exp1 -lt exp2	True, if exp1 is less than exp2.
exp1 -gt exp2	True, if exp1 is greater than exp2.
exp1 -le exp2	True, if exp1 is less than or equal to exp2.
exp1 -ge exp2	True, if exp1 is greater than or equal to exp2.

In each of the above expressions, if file is of the form /dev/fd/n, where n is an integer, then the test is applied to the open file whose descriptor number is n.

A compound expression can be constructed from these primitives by using any of the following, listed in decreasing order of precedence.

(expression)	True, if expression is true. Used to group expressions.
! expression	True if expression is false.
expression1 && expression2	True, if expression1 and expression2 are both true.
expression1 || expression2	True, if either expression1 or expression2 is true.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a command and are not passed on to the invoked command. Command and parameter substitution occur before word or digit is used except as noted below. File name generation occurs only if the pattern matches a single file, and blank interpretation is not performed.

<word	Use file word as standard input (file descriptor 0).
>word	Use file word as standard output (file descriptor 1). If the file does not exist then it is created. If the file exists, and the noclobber option is on, this causes an error; otherwise, it is truncated to zero length.
>|word	Sames as >, except that it overrides the noclobber option.
>>word	Use file word as standard output. If the file exists then output is appended to it (by first seeking to the EOF); otherwise, the file is created.
<>word	Open file word for reading and writing as

standard input.

<< [-]word — The shell input is read up to a line that is the same as word, or to an EOF. No parameter substitution, command substitution or file name generation is performed on word. The resulting document, called a here-document, becomes the standard input. If any character of word is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occur, \NEWLINE is ignored, and \ must be used to quote the characters \, $, `, and the first character of word. If - is appended to <<, then all leading tabs are stripped from word and from the document.

<&digit — The standard input is duplicated from file descriptor digit (see dup(2)). Similarly for the standard output using >&digit.

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

<&p — The input from the co-process is moved to standard input.

>&p — The output to the co-process is moved to standard output.

If one of the above is preceded by a digit, then the file descriptor number referred to is that specified by the digit (instead of the default 0 or 1). For example:

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

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.
The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (file descriptor, file) association at the time of evaluation. For example:

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

first associates file descriptor 1 with file fname. It then associates file descriptor 2 with the file associated with

file descriptor 1 (that is fname). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be associated with file fname.

If a command is followed by & and job control is not active, then the default standard input for the command is the empty file /dev/null. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

ENVIRONMENT

The environment (see environ(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The names must be identifiers and the values are character strings. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a variable for each name found, giving it the corresponding value and marking it export. Executed commands inherit the environment. If the user modifies the values of these variables or creates new ones, using the export or typeset -x commands they become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions which must be noted in export or typeset -x commands.

The environment for any simple-command or function may be augmented by prefixing it with one or more variable assignments. A variable assignment argument is a word of the form identifier=value. Thus:

```
TERM=450 cmd args
and
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of cmd is concerned except for special commands listed below that are preceded with a dagger).

If the -k flag is set, all variable assignment arguments are placed in the environment, even if they occur after the command name. The following first prints a=b c and then c:

```
echo a=b c
```

```
set -k
echo a=b c
```

This feature is intended for use with scripts written for early versions of the shell and its use in new scripts is strongly discouraged. It is likely to disappear someday.

Functions

The function reserved word, described in the Commands section above, is used to define shell functions. Shell functions are read in and stored internally. Alias names are resolved when the function is read. Functions are executed like commands with the arguments passed as positional parameters. (See Execution below.)

Functions execute in the same process as the caller and share all files and present working directory with the caller. Traps caught by the caller are reset to their default action inside the function. A trap condition that is not caught or ignored by the function causes the function to terminate and the condition to be passed on to the caller. A trap on EXIT set inside a function is executed after the function completes in the environment of the caller. Ordinarily, variables are shared between the calling program and the function. However, the typeset special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command return is used to return from function calls. Errors within functions return control to the caller.

The names of all functions can be listed with typeset +f. typeset -f lists all function names as well as the text of all functions. typeset -f function-names lists the text of the named functions only. Functions can be undefined with the -f option of the unset special command.

Ordinarily, functions are unset when the shell executes a shell script. The -xf option of the typeset command allows a function to be exported to scripts that are executed without a separate invocation of the shell. Functions that need to be defined across separate invocations of the shell should be specified in the ENV file with the -xf option of typeset.

Function Definition Command

A function is a user-defined name that is used as a simple command to call a compound command with new positional parameters. A function is defined with a function definition command.

The format of a function definition command is as follows:

```
fname() compound-command[io-redirect ...]
```

The function is named fname; it must be a name. An implementation may allow other characters in a function name as an extension. The implementation will maintain separate name spaces for functions and variables.

The () in the function definition command consists of two operators. Therefore, intermixing blank characters with the fname, (, and) is allowed, but unnecessary.

The argument compound-command represents a compound command.

When the function is declared, none of the expansions in wordexp will be performed on the text in compound-command or io-redirect; all expansions will be performed as normal each time the function is called. Similarly, the optional io-redirect redirections and any variable assignments within compound-command will be performed during the execution of the function itself, not the function definition.

When a function is executed, it will have the syntax-error and variable-assignment properties described for the special built-in utilities.

The compound-command will be executed whenever the function name is specified as the name of a simple command The operands to the command temporarily will become the positional parameters during the execution of the compound-command; the special parameter # will also be changed to reflect the number of operands. The special parameter 0 will be unchanged. When the function completes, the values of the positional parameters and the special parameter # will be restored to the values they had before the function was executed. If the special built-in return is executed in the compound-command, the function will complete and execution will resume with the next command after the function call.

An example of how a function definition can be used wherever a simple command is allowed:

```
# If variable i is equal to "yes",
# define function foo to be ls -l
#
[ "$i" = yes ] && foo() {
        ls -l
}
```

The exit status of a function definition will be 0 if the function was declared successfully; otherwise, it will be greater than zero. The exit status of a function invocation will be the exit status of the last command executed by the function.

Jobs

If the monitor option of the set command is turned on, an interactive shell associates a job with each pipeline. It keeps a table of current jobs, printed by the jobs command, and assigns them small integer numbers. When a job is started asynchronously with &, the shell prints a line which looks like:

```
[1] 1234
```

indicating that the job, 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 the key ^Z (CTRL-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been `Stopped', and print another prompt. You can then manipulate the state of this job, putting it in the background with the bg command, or run some other commands and then eventually bring the job back into the foreground with the foreground command fg. A ^Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

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 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. A job can be referred to by the process id of any process of the job or by one of the following:

%number	The job with the given number.
%string	Any job whose command line begins with string.
%?string	Any job whose command line contains string.
%%	Current job.
%+	Equivalent to %%.
%-	Previous job.

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

When the monitor mode is on, each background job that completes triggers any trap set for CHLD.

When you try to leave the shell while jobs are running or stopped, you will be warned that `You have stopped(running) 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 stopped jobs will be terminated. If you have nohup'ed jobs running when you attempt to logout, you will be warned with the message

You have jobs running.

You will then need to logout a second time to actually logout; however, your background jobs will continue to run.

Signals

The INT and QUIT signals for an invoked command are ignored if the command is followed by & and the monitor option is not active. Otherwise, signals have the values inherited by the shell from its parent (but see also the trap special command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the Special Commands listed below, it is executed within the current shell process. Next, the command name is checked to see if it matches one of the user defined functions. If it does,

the positional parameters are saved and then reset to the arguments of the function call. When the function completes or issues a return, the positional parameter list is restored and any trap set on EXIT within the function is executed. The value of a function is the value of the last command executed. A function is also executed in the current shell process. If a command name is not a special command or a user defined function, a process is created and an attempt is made to execute the command via exec(2).

The shell variable PATH defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is /bin:/usr/bin: (specifying /bin, /usr/bin, and the current directory in that order). The current directory can be specified by two or more adjacent colons, or by a colon at the beginning or end of the path list. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an a.out file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. All non-exported aliases, functions, and variables are removed in this case. A parenthesized command is executed in a sub-shell without removing non-exported quantities.

Command Re-entry

The text of the last HISTSIZE (default 128) commands entered from a terminal device is saved in a history file. The file $HOME/.sh_history is used if the HISTFILE variable is not set or if the file it names is not writable. A shell can access the commands of all interactive shells which use the same named HISTFILE. The special command fc is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified. If you do not specify an editor program as an argument to fc then the value of the variable FCEDIT is used. If FCEDIT is not defined then /bin/ed is used. The edited command(s) is printed and re-executed upon leaving the editor. The editor name - is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form old=new can be used to modify the command before execution. For example, if r is aliased to 'fc -e -' then typing `r bad=good c' will re-execute the most recent command which starts with the letter c, replacing the first occurrence of

the string bad with the string good.

In-line Editing Option

Normally, each command line entered from a terminal device is simply typed followed by a new-line (RETURN or LINEFEED). If either the emacs, gmacs, or vi option is active, the user can edit the command line. To be in either of these edit modes set the corresponding option. An editing option is automatically selected each time the VISUAL or EDITOR variable is assigned a value ending in either of these option names.

The editing features require that the user's terminal accept RETURN as carriage return without line feed and that a space must overwrite the current character on the screen.

The editing modes implement a concept where the user is looking through a window at the current line. The window width is the value of COLUMNS if it is defined, otherwise 80. If the window width is too small to display the prompt and leave at least 8 columns to enter input, the prompt is truncated from the left. If the line is longer than the window width minus two, a mark is displayed at the end of the window to notify the user. As the cursor moves and reaches the window boundaries the window will be centered about the cursor. The mark is a > if the line extends on the right side of the window, < if the line extends on the left, and * if the line extends on both sides of the window.

The search commands in each edit mode provide access to the history file. Only strings are matched, not patterns, although a leading ^ in the string restricts the match to begin at the first character in the line.

emacs Editing Mode

This mode is entered by enabling either the emacs or gmacs option. The only difference between these two modes is the way they handle ^T. To edit, the user moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. All the editing commands are control characters or escape sequences. The notation for control characters is caret (^) followed by the character. For example, ^F is the notation for control F. This is entered by depressing `f' while holding down the CTRL (control) key. The SHIFT key is not depressed. (The notation ^? indicates the DEL (delete) key.)

The notation for escape sequences is M- followed by a character. For example, M-f (pronounced Meta f) is entered by depressing ESC (ascii 033) followed by `f'. (M-F would be the notation for ESC followed by SHIFT (capital) `F'.)

All edit commands operate from any place on the line (not just at the beginning). Neither the RETURN nor the LINEFEED key is entered after edit commands except when noted.

^F	Move cursor forward (right) one character.
M-f	Move cursor forward one word. (The emacs editor's idea of a word is a string of characters consisting of only letters, digits and underscores.)
^B	Move cursor backward (left) one character.
M-b	Move cursor backward one word.
^A	Move cursor to start of line.
^E	Move cursor to end of line.
^]char	Move cursor forward to character char on current line.
M-^]char	Move cursor backward to character char on current line.
^X^X	Interchange the cursor and mark.
erase	(User defined erase character as defined by the stty(1) command, usually ^H or #.) Delete previous character.
^D	Delete current character.
M-d	Delete current word.
M-^H	(Meta-backspace) Delete previous word.
M-h	Delete previous word.
M-^?	(Meta-DEL) Delete previous word (if your interrupt character is ^? (DEL, the default) then this command will not work).
^T	Transpose current character with next character in emacs mode. Transpose two previous characters in gmacs mode.
^C	Capitalize current character.
M-c	Capitalize current word.
M-l	Change the current word to lower case.
^K	Delete from the cursor to the end of the line. If preceded by a numerical parameter whose value is less than the current cursor position, then delete from given position up to the cursor. If preceded by a numerical parameter whose value is greater than the current cursor position, then delete from cursor up to given cursor position.

^W	Kill from the cursor to the mark.
M-p	Push the region from the cursor to the mark on the stack.
kill	(User defined kill character as defined by the stty(1) command, usually ^G or @.) Kill the entire current line. If two kill characters are entered in succession, all kill characters from then on cause a line feed (useful when using paper terminals).
^Y	Restore last item removed from line. (Yank item back to the line.)
^L	Line feed and print current line.
^@	(null character) Set mark.
M-space	(Meta space) Set mark.
J	(New line) Execute the current line.
M	(Return) Execute the current line.
eof	End-of-file character, normally ^D, is processed as an End-of-file only if the current line is null.
^P	Fetch previous command. Each time ^P is entered the previous command back in time is accessed. Moves back one line when not on the first line of a multi-line command.
M-<	Fetch the least recent (oldest) history line.
M->	Fetch the most recent (youngest) history line.
^N	Fetch next command line. Each time ^N is entered the next command line forward in time is accessed.
^Rstring	Reverse search history for a previous command line containing string. If a parameter of zero is given, the search is forward. string is terminated by a RETURN or NEW LINE. If string is preceded by a ^, the matched line must begin with string. If string is omitted, then the next command line containing the most recent string is accessed. In this case a parameter of zero reverses the direction of the search.
^O	Operate. Execute the current line and fetch the next line relative to current line from the history file.
M-digits	(Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a parameter are ^F, ^B, erase, ^C, ^D, ^K, ^R, ^P, ^N, ^], M-., M-^], M-_, M-b, M-c, M-d, M-f, M-h, M-l and

M-^H.

M-letter	Soft-key. Your alias list is searched for an alias by the name _letter and if an alias of this name is defined, its value will be inserted on the input queue. The letter must not be one of the above meta-functions.
M-[letter	Soft-key. Your alias list is searched for an alias by the name __letter and if an alias of this name is defined, its value will be inserted on the input queue. The can be used to program functions keys on many terminals.
M-.	The last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.
M-_	Same as M-..
M-*	An asterisk is appended to the end of the word and a file name expansion is attempted.
M-ESC	File name completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.
M-=	List files matching current word pattern if an asterisk were appended.
^U	Multiply parameter of next command by 4.
\	Escape next character. Editing characters, the user's erase, kill and interrupt (normally ^?) characters may be entered in a command line or in a search string if preceded by a \. The \ removes the next character's editing features (if any).
^V	Display version of the shell.
M-#	Insert a # at the beginning of the line and execute it. This causes a comment to be inserted in the history file.

vi Editing Mode

There are two typing modes. Initially, when you enter a command you are in the input mode. To edit, the user enters control mode by typing ESC (033) and moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. Most control commands accept an optional repeat count prior to the command.

When in vi mode on most systems, canonical processing is initially enabled and the command will be echoed again if the speed is 1200 baud or greater and it contains any control characters or less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command and the user can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.

If the option viraw is also set, the terminal will always have canonical processing disabled. This mode is implicit for systems that do not support two alternate end of line delimiters, and may be helpful for certain terminals.

Input Edit Commands

By default the editor is in input mode.

erase	(User defined erase character as defined by the stty(1) command, usually ^H or #.) Delete previous character.
^W	Delete the previous blank separated word.
^D	Terminate the shell.
^V	Escape next character. Editing characters and the user's erase or kill characters may be entered in a command line or in a search string if preceded by a ^V. The ^V removes the next character's editing features (if any).
\	Escape the next erase or kill character.

Motion Edit Commands

These commands will move the cursor.

[count]l	Cursor forward (right) one character.
[count]w	Cursor forward one alpha-numeric word.
[count]W	Cursor to the beginning of the next word that follows a blank.
[count]e	Cursor to end of word.

[count]E Cursor to end of the current blank delimited word.

[count]h Cursor backward (left) one character.

[count]b Cursor backward one word.

[count]B Cursor to preceding blank separated word.

[count]| Cursor to column count.

[count]fc Find the next character c in the current line.

[count]Fc Find the previous character c in the current line.

[count]tc Equivalent to f followed by h.

[count]Tc Equivalent to F followed by l.

[count]; Repeats count times, the last single character find command, f, F, t, or T.

[count], Reverses the last single character find command count times.

0 Cursor to start of line.

^ Cursor to first non-blank character in line.

$ Cursor to end of line.

% Moves to balancing (,), {, }, [, or]. If cursor is not on one of the above characters, the remainder of the line is searched for the first occurrence of one of the above characters first.

Search Edit Commands

These commands access your command history.

[count]k Fetch previous command. Each time k is entered the previous command back in time is accessed.

[count]- Equivalent to k.

[count]j Fetch next command. Each time j is entered the next command forward in time is accessed.

[count]+ Equivalent to j.

[count]G The command number count is fetched. The default is the least recent history command.

/string Search backward through history for a previous command containing string. string is terminated by a RETURN or NEWLINE. If string is preceded by a ^, the matched line must begin with string. If string is NULL, the previous string will be used.

?string Same as / except that search will be in the forward direction.

n Search for next match of the last pattern to / or ? commands.

N Search for next match of the last pattern to / or ?, but in reverse direction. Search history for the string entered by the previous / command.

Text Modification Edit Commands
These commands will modify the line.

a Enter input mode and enter text after the current character.

A Append text to the end of the line. Equivalent to $a.

[count]cmotion
c[count]motion
Delete current character through the character that motion would move the cursor to and enter input mode. If motion is c, the entire line will be deleted and input mode entered.

C Delete the current character through the end of line and enter input mode. Equivalent to c$.

[count]s Delete count characters and enter input mode.

S Equivalent to cc.

D Delete the current character through the end of line. Equivalent to d$.

[count]dmotion
d[count]motion
Delete current character through the character that motion would move to. If motion is d, the entire line will be deleted.

i Enter input mode and insert text before the current character.

I Insert text before the beginning of the line. Equivalent to 0i.

[count]P Place the previous text modification before the cursor.

[count]p Place the previous text modification after the cursor.

R Enter input mode and replace characters on the screen with characters you type overlay fashion.

[count]rc Replace the count character(s) starting at the current cursor position with c, and advance the cursor.

[count]x Delete current character.

[count]X Delete preceding character.

[count]. Repeat the previous text modification command.

[count]~ Invert the case of the count character(s) starting at the current cursor position and advance the cursor.

[count]_ Causes the count word of the previous command to be appended and input mode entered. The last word is used if count is omitted.

* Causes an * to be appended to the current

word and file name generation attempted. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.

\ Filename completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.

Other Edit Commands

Miscellaneous commands.

[count]ymotion

y[count]motion

Yank current character through character that motion would move the cursor to and puts them into the delete buffer. The text and cursor are unchanged.

Y Yanks from current position to end of line. Equivalent to y$.

u Undo the last text modifying command.

U Undo all the text modifying commands performed on the line.

[count]v

Returns the command fc -e ${VISUAL:-${EDITOR:-vi}} count in the input buffer. If count is omitted, then the current line is used.

^L Line feed and print current line. Has effect only in control mode.

J (New line) Execute the current line, regardless of mode.

M (Return) Execute the current line, regardless of mode.

If the first character of the command is a #, then this command deletes this # and each # that follows a newline. Otherwise, sends the line after

inserting a # in front of each line in the command. Useful for causing the current line to be inserted in the history as a comment and removing comments from previous comment commands in the history file.

= List the file names that match the current word if an asterisk were appended it.

@letter
Your alias list is searched for an alias by the name _letter and if an alias of this name is defined, its value will be inserted on the input queue for processing.

Special Commands

The following simple-commands are executed in the shell process. Input/Output redirection is permitted. Unless otherwise indicated, the output is written on file descriptor 1 and the exit status, when there is no syntax error, is 0. Commands that are preceded by one or two * (asterisks) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by ** that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the = sign and word splitting and file name generation are not performed.

* : [arg ...]
The command only expands parameters.

* . file [arg ...]
Read the complete file then execute the commands. The commands are executed in the current shell environment. The search path specified by PATH is used to find the directory containing file. If any arguments arg are given, they become the positional parameters. Otherwise the positional parameters are unchanged. The exit status is the exit status of the last command executed.

** alias [-tx] [name[=value]] ...

alias with no arguments prints the list of aliases in the form name=value on standard output. An alias is defined for each name whose value is given. A trailing space in value causes the next word to be checked for alias substitution. The -t flag is used to set and list tracked aliases. The value of a tracked alias is the full pathname corresponding to the given name. The value becomes undefined when the value of PATH is reset but the aliases remained tracked. Without the -t flag, for each name in the argument list for which no value is given, the name and value of the alias is printed. The -x flag is used to set or print exported aliases. An exported alias is defined for scripts invoked by name. The exit status is non-zero if a name is given, but no value, and no alias has been defined for the name.

bg [%job...]
This command is only on systems that support job control. Puts each specified job into the background. The current job is put in the background if job is not specified. See "Jobs" section above for a description of the format of job.

* break [n]
Exit from the enclosed for, while, until, or select loop, if any. If n is specified then break n levels.

* continue [n]
Resume the next iteration of the enclosed for, while, until, or select loop. If n is specified then resume at the n-th enclosed loop.

cd [arg]
cd old new
This command can be in either of two forms. In the first form it changes the current directory to arg. If arg is - the directory is changed to the previous directory. The shell variable HOME is the default arg. The variable PWD is set to the current directory. The shell variable CDPATH defines the search path for the directory containing arg. Alternative directory names are separated by a colon (:). The default path is null (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path

list. If arg begins with a / then the search path is not used. Otherwise, each directory in the path is searched for arg.

The second form of cd substitutes the string new for the string old in the current directory name, PWD and tries to change to this new directory.

The cd command may not be executed by rksh.

command [-p] [command_name] [argument ...]
command [-v -V] command_name

The command utility causes the shell to treat the arguments as a simple command, suppressing the shell function lookup. The -p flag performs the command search using a default value for PATH that is guaranteed to find all of the standard utilities. The -v flag writes a string to standard output that indicates the pathname or command that will be used by the shell, in the current shell execution environment, to invoke command_name. The -V flag writes a string to standard output that indicates how the name given in the command_name operand will be interpreted by the shell, in the current shell execution environment.

* eval [arg ...]

The arguments are read as input to the shell and the resulting command(s) executed.

* exec [arg ...]

If arg is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and affect the current process. If no arguments are given the effect of this command is to modify file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 that are opened with this mechanism are closed when invoking another program.

* exit [n]

Causes the calling shell or shell script to exit with the exit status specified by n. The value will be the least significant 8 bits of the specified status. If n is omitted then the exit status is that of the last command executed. When exit occurs when executing a trap, the last command refers to the command that executed before the trap was invoked. An EOF will also cause the shell to exit except for a shell which has

the ignoreeof option (See set below) turned on.

** export [name[=value]] ...
The given names are marked for automatic export to the environment of subsequently-executed commands.

fc [-e ename] [-nlr] [first [last]]
fc -e - [old=new] [command]
In the first form, a range of commands from first to last is selected from the last HISTSIZE commands that were typed at the terminal. The arguments first and last may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the -l flag is selected, the commands are listed on standard output. Otherwise, the editor program ename is invoked on a file containing these keyboard commands. If ename is not supplied, then the value of the variable FCEDIT (default /bin/ed) is used as the editor. When editing is complete, the edited command(s) is executed. If last is not specified then it will be set to first. If first is not specified the default is the previous command for editing and -16 for listing. The flag -r reverses the order of the commands and the flag -n suppresses command numbers when listing. In the second form the command is re-executed after the substitution old=new is performed. If there is not a command argument, the most recent command typed at this terminal is executed.

fg [%job...]
This command is only on systems that support job control. Each job specified is brought to the foreground. Otherwise, the current job is brought into the foreground. See "Jobs" section above for a description of the format of job.

getopts optstring name [arg ...]
Checks arg for legal options. If arg is omitted, the positional parameters are used. An option argument begins with a + or a -. An option not beginning with + or - or the argument -- ends the options. optstring contains the letters that getopts recognizes. If a letter is followed by a :, that option is expected to have an argument. The options can be separated from the argument by blanks.

getopts places the next option letter it finds inside variable name each time it is invoked with a + prepended when arg begins with a +. The index of the next arg is stored in OPTIND. The option argument, if any, gets stored in OPTARG.

A leading : in optstring causes getopts to store the letter of an invalid option in OPTARG, and to set name to ? for an unknown option and to : when a required option is missing. Otherwise, getopts prints an error message. The exit status is non-zero when there are no more options. See getoptcvt(1) for usage and description.

hash [name ...]
For each name, the location in the search path of the command specified by name is determined and remembered by the shell. The -r option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. Hits is the number of times a command has been invoked by the shell process. Cost is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the hits information. Cost will be incremented when the recalculation is done.

jobs [-lnp] [%job ...]
Lists information about each given job; or all active jobs if job is omitted. The -l flag lists process ids in addition to the normal information. The -n flag displays only jobs that have stopped or exited since last notified. The -p flag causes only the process group to be listed. See "Jobs" section above and jobs(1) for a description of the format of job.

kill [-sig] %job ...
kill [-sig] pid ...
kill -l
Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in signal(5) stripped of the prefix ``SIG'' with the

exception that SIGCHD is named CHLD). If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal if it is stopped. The argument job can be the process id of a process that is not a member of one of the active jobs. See Jobs for a description of the format of job. In the second form, kill -l, the signal numbers and names are listed.

let arg...
Each arg is a separate arithmetic expression to be evaluated. See the Arithmetic Evaluation section above, for a description of arithmetic expression evaluation.

The exit status is 0 if the value of the last expression is non-zero, and 1 otherwise.

login argument ...
Equivalent to `exec login argument....' See login(1) for usage and description.

* newgrp [arg ...]
Equivalent to exec /bin/newgrp arg ....

print [-Rnprsu[n]] [arg ...]
The shell output mechanism. With no flags or with flag - or --, the arguments are printed on standard output as described by echo(1). The exit status is 0, unless the output file is not open for writing.

-n Suppress NEWLINE from being added to the output.

-R | -r Raw mode. Ignore the escape conventions of echo. The -R option will print all subsequent arguments and options other than -n.

-p Write the arguments to the pipe of the process spawned with |& instead of standard output.

-s Write the arguments to the history file instead of standard output.

-u [n] Specify a one digit file descriptor unit number n on which the output will be placed. The default is 1.

pwd Equivalent to print -r - $PWD.

mechanism. One line is read and
read [-prsu[n]] [name?prompt] [name ...] The shell input
is broken up into fields using the characters in IFS as separators. The escape character, (\), is used to remove any special meaning for the next character and for line continuation. In raw mode, -r, the \ character is not treated specially. The first field is assigned to the first name, the second field to the second name, etc., with leftover fields assigned to the last name. The -p option causes the input line to be taken from the input pipe of a process spawned by the shell using |&. If the -s flag is present, the input will be saved as a command in the history file. The flag -u can be used to specify a one digit file descriptor unit n to read from. The file descriptor can be opened with the exec special command. The default value of n is 0. If name is omitted then REPLY is used as the default name. The exit status is 0 unless the input file is not open for reading or an EOF is encountered. An EOF with the -p option causes cleanup for this process so that another can be spawned. If the first argument contains a ?, the remainder of this word is used as a prompt on standard error when the shell is interactive. The exit status is 0 unless an EOF is encountered.

** readonly [name[=value]] ...
The given names are marked readonly and these names cannot be changed by subsequent assignment.

* return [n]
Causes a shell function or '.' script to return to the invoking script with the return status specified by n. The value will be the least significant 8 bits of the specified status. If n is omitted then the return status is that of the last command executed. If return is invoked while not in a function or a '.' script, then it is the same as an exit.
set [+-abCefhkmnopstuvx] [+-o option]...
[+-A name] [arg ...]
The flags for this command have meaning as follows:

-A Array assignment. Unset the variable name and

assign values sequentially from the list arg. If +A is used, the variable name is not unset first.

-a All subsequent variables that are defined are automatically exported.

-b Causes the shell to notify the user asynchronously of background job completions. The following message will be written to standard error:

"[%d]%c %s%s\n", <job-number>, <current>, <status>, <job-name>

where the fields are as follows:

<current> The character + identifies the job that would be used as a default for the fg or bg utilities; this job can also be specified using the job_id %+ or %%. The character - identifies the job that would become the default if the current default job were to exit; this job can also be specified using the job_id %-. For other jobs, this field is a space character. At most one job can be identified with + and at most one job can be identified with -. If there is any suspended job, then the current job will be a suspended job. If there are at least two suspended jobs, then the previous job will also be a suspended job.

<job-number> A number that can be used to identify the process group to the wait, fg, bg, and kill utilities. Using these utilities, the job can be identified by prefixing the job number with %.

<status> Unspecified.

<job-name> Unspecified.

When the shell notifies the user a job has been

completed, it may remove the job's process ID from the list of those known in the current shell execution environment. Asynchronous notification will not be enabled by default.

-C Prevent existing files from being overwritten by the shell's > redirection operator; the >| redirection operator will override this noclobber option for an individual file.

-e If a command has a non-zero exit status, execute the ERR trap, if set, and exit. This mode is disabled while reading profiles.

-f Disables file name generation.

-h Each command becomes a tracked alias when first encountered.

-k All variable assignment arguments are placed in the environment for a command, not just those that precede the command name.

-m Background jobs will run in a separate process group and a line will print upon completion. The exit status of background jobs is reported in a completion message. On systems with job control, this flag is turned on automatically for interactive shells.

-n Read commands and check them for syntax errors, but do not execute them. Ignored for interactive shells.

-o The following argument can be one of the following option names:

allexport	Same as -a.
errexit	Same as -e.
bgnice	All background jobs are run at a lower priority. This is the default mode.
emacs	Puts you in an emacs style in-line editor for command entry.
gmacs	Puts you in a gmacs style in-line editor for command entry.
ignoreeof	The shell will not exit on EOF. The command exit must be used.
keyword	Same as -k.

markdirs All directory names resulting from file name generation have a trailing / appended.
monitor Same as -m.
noclobber Prevents redirection > from truncating existing files. Require >| to truncate a file when turned on. Equivalent to -C.
noexec Same as -n.
noglob Same as -f.
nolog Do not save function definitions in history file.
notify Equivalent to -b.
nounset Same as -u.
privileged Same as -p.
verbose Same as -v.
trackall Same as -h.
vi Puts you in insert mode of a vi style in-line editor until you hit escape character 033. This puts you in control mode. A return sends the line.
viraw Each character is processed as it is typed in vi mode.
xtrace Same as -x.

If no option name is supplied then the current option settings are printed.

-p Disables processing of the $HOME/.profile file and uses the file /etc/suid_profile instead of the ENV file. This mode is on whenever the effective uid is not equal to the real uid, or when the effective gid is not equal to the real gid. Turning this off causes the effective uid and gid to be set to the real uid and gid.

-s Sort the positional parameters lexicographically.

-t Exit after reading and executing one command.

-u Treat unset parameters as an error when substituting.

-v Print shell input lines as they are read.

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

- Turns off -x and -v flags and stops examining arguments for flags.

-- Do not change any of the flags; useful in setting $1 to a value beginning with -. If no arguments follow this flag then the positional parameters are unset.

Using + rather than - causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in $-. Unless -A is specified, the remaining arguments are positional parameters and are assigned, in order, to $1 $2 .... If no arguments are given then the names and values of all variables are printed on the standard output.

* shift [n]

The positional parameters from $n+1 $n+1 ... are renamed $1 ..., default n is 1. The parameter n can be any arithmetic expression that evaluates to a non-negative number less than or equal to $#.

stop %jobid ...
stop pid ...

stop stops the execution of a background job(s) by using its jobid, or of any process by using its pid. (see ps(1)).

suspend

Stops the execution of the current shell (but not if it is the login shell).

test expression

Evaluate conditional expressions. See Conditional Expressions section above and test(1) for usage and description.

* times

Print the accumulated user and system times for the shell and for processes run from the shell.

* trap [arg sig ...]

arg is a command to be read and executed when the shell receives signal(s) sig. arg is scanned once when the trap is set and once when the trap is taken. sig can be specified as a signal number or signal name. trap com-

mands are executed in order of signal number. Any attempt to set a trap on a signal number that was ignored on entry to the current shell is ineffective.

If arg is -, the shell will reset each sig to the default value. If arg is null (''), the shell will ignore each specified sig if it arises. Otherwise, arg will be read and executed by the shell when one of the corresponding sigs arises. The action of the trap will override a previous action (either default action or one explicitly set). The value of $? after the trap action completes will be the value it had before the trap was invoked.
sig can be EXIT, 0 (equivalent to EXIT) or a signal specified using a symbolic name, without the SIG prefix, for example, HUP, INT, QUIT, TERM. If sig is 0 or EXIT and the trap statement is executed inside the body of a function, then the command arg is executed after the function completes. If sig is 0 or EXIT for a trap set outside any function then the command arg is executed on exit from the shell. If sig is ERR then arg will be executed whenever a command has a non-zero exit status. If sig is DEBUG then arg will be executed after each command.

The environment in which the shell executes a trap on EXIT will be identical to the environment immediately after the last command executed before the trap on EXIT was taken.

Each time the trap is invoked, arg will be processed in a manner equivalent to:

```
eval "$arg"
```

Signals that were ignored on entry to a non-interactive shell cannot be trapped or reset, although no error need be reported when attempting to do so. An interactive shell may reset or catch signals ignored on entry. Traps will remain in place for a given shell until explicitly changed with another trap command.

When a subshell is entered, traps are set to the default args. This does not imply that the trap command cannot be used within the subshell to set new traps.

The trap command with no arguments will write to standard output a list of commands associated with each sig. The format is:

```
trap -- %s %s ... <arg>, <sig> ...
```

The shell will format the output, including the proper use of quoting, so that it is suitable for reinput to the shell as commands that achieve the same trapping results. For example:

```
save_traps=$(trap)
...
eval "$save_traps"
```

If the trap name or number is invalid, a non-zero exit status will be returned; otherwise, 0 will be returned. For both interactive and non-interactive shells, invalid signal names or numbers will not be considered a syntax error and will not cause the shell to abort.

Traps are not processed while a job is waiting for a foreground process. Thus, a trap on CHLD won't be executed until the foreground job terminates.

type name ...
For each name, indicate how it would be interpreted if used as a command name.

** typeset [+HLRZfilrtux[n]] [name[=value]] ...
Sets attributes and values for shell variables and functions. When typeset is invoked inside a function, a new instance of the variables name is created. The variables value and type are restored when the function completes. The following list of attributes may be specified:

-H This flag provides UNIX to host-name file mapping on non-UNIX machines.

-L Left justify and remove leading blanks from value. If n is non-zero it defines the width of the field; otherwise, it is determined by the width of the value of first assignment. When the variable is assigned to, it is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the -Z flag is also set. The -R flag is turned off.

-R Right justify and fill with leading blanks. If n is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. The field is left filled with blanks or truncated from the end if the variable is reassigned. The -L flag is turned off.

-Z Right justify and fill with leading zeros if the first non-blank character is a digit and the -L flag has not been set. If n is non-zero it defines the width of the field; otherwise, it is determined by the width of the value of first assignment.

-f The names refer to function names rather than variable names. No assignments can be made and the only other valid flags are -t, -u and -x. The flag -t turns on execution tracing for this function. The flag -u causes this function to be marked undefined. The FPATH variable will be searched to find the function definition when the function is referenced. The flag -x allows the function definition to remain in effect across shell procedures invoked by name.

-i Parameter is an integer. This makes arithmetic faster. If n is non-zero it defines the output arithmetic base; otherwise, the first assignment determines the output base.

-l All upper-case characters are converted to lower-case. The upper-case flag, -u is turned off.

-r The given names are marked readonly and these names cannot be changed by subsequent assignment.

-t Tags the variables. Tags are user definable and have no special meaning to the shell.

-u All lower-case characters are converted to upper-case characters. The lower-case flag, -l is turned off.

-x The given names are marked for automatic export to the environment of subsequently-executed commands.

The -i attribute can not be specified along with -R, -L, -Z, or -f.

Using + rather than - causes these flags to be turned off. If no name arguments are given but flags are specified, a list of names (and optionally the values) of the variables which have these flags set is printed. (Using + rather than - keeps the values from being

printed.) If no names and flags are given, the names and attributes of all variables are printed.

ulimit [-HSacdfnstv] [limit]

Set or display a resource limit. The available resources limits are listed below. Many systems do not contain one or more of these limits. The limit for a specified resource is set when limit is specified. The value of limit can be a number in the unit specified below with each resource, or the value unlimited. The H and S flags specify whether the hard limit or the soft limit for the given resource is set. A hard limit cannot be increased once it is set. A soft limit can be increased up to the value of the hard limit. If neither the H or S options is specified, the limit applies to both. The current resource limit is printed when limit is omitted. In this case the soft limit is printed unless H is specified. When more that one resource is specified, then the limit name and unit is printed before the value.

-a Lists all of the current resource limits.

-c The number of 512-byte blocks on the size of core dumps.

-d The number of K-bytes on the size of the data area.

-f The number of 512-byte blocks on files written by child processes (files of any size may be read).

-n The number of file descriptors plus 1.

-s The number of K-bytes on the size of the stack area.

-t The number of seconds to be used by each process.

-v The number of K-bytes for virtual memory.

If no option is given, -f is assumed.

umask [-S] [mask]

The user file-creation mask is set to mask (see umask(2)). mask can either be an octal number or a symbolic value as described in chmod(1). If a symbolic value is given, the new umask value is the complement of the result of applying mask to the complement of the previous umask value. If mask is omitted, the current value of the mask is printed. The -S flag produces symbolic output.

unalias name...

The aliases given by the list of names are removed from

the alias list.

unset [-f] name ...
The variables given by the list of names are unassigned, that is, their values and attributes are erased. readonly variables cannot be unset. If the -f, flag is set, then the names refer to function names. Unsetting ERRNO, LINENO, MAILCHECK, OPTARG, OPTIND, RANDOM, SECONDS, TMOUT, and _ removes their special meaning even if they are subsequently assigned to.

* wait [job]
Wait for the specified job and report its termination status. If job is not given then all currently active child processes are waited for. The exit status from this command is that of the process waited for. See Jobs for a description of the format of job.

whence [-pv] name ...
For each name, indicate how it would be interpreted if used as a command name.

The -v flag produces a more verbose report.

The -p flag does a path search for name even if name is an alias, a function, or a reserved word.

Invocation

If the shell is invoked by exec(2), and the first character of argument zero ($0) is -, then the shell is assumed to be a login shell and commands are read from /etc/profile and then from either .profile in the current directory or $HOME/.profile, if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment variable ENV if the file exists. If the -s flag is not present and arg is, then a path search is performed on the first arg to determine the name of the script to execute. The script arg must have read permission and any setuid and setgid settings will be ignored. If the script is not found on the path, arg is processed as if it named a builtin command or function. Commands are then read as described below; the following flags are interpreted by the shell when it is invoked:

-c string If the -c flag is present then commands are read from string.

-s If the -s flag is present or if no arguments remain then commands are read from the standard input. Shell output, except for the output of the Special Commands listed above, is written to file descriptor 2.

-i If the -i flag is present or if the shell input and output are attached to a terminal (as told by ioctl(2)) then this shell is interactive. In this case TERM is ignored (so that kill 0 does not kill an interactive shell) and INTR is caught and ignored (so that wait is interruptible). In all cases, QUIT is ignored by the shell.

-r If the -r flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the set command above.

rksh Only

rksh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of rksh are identical to those of ksh, except that the following are disallowed:

+ changing directory (see cd(1))
+ setting the value of SHELL, ENV, or PATH
+ specifying path or command names containing /
+ redirecting output (>, >|, <>, and >>)
+ changing group (see newgrp(1)).

The restrictions above are enforced after .profile and the ENV files are interpreted.

When a command to be executed is found to be a shell procedure, rksh invokes ksh to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the .profile has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably not the login directory).

The system administrator often sets up a directory of commands (that is, /usr/rbin) that can be safely invoked by

rksh.

ERRORS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. Otherwise, the shell returns the exit status of the last command executed (see also the exit command above). If the shell is being used non-interactively then execution of the shell file is abandoned. Run time errors detected by the shell are reported by printing the command or function name and the error condition. If the line number that the error occurred on is greater than one, then the line number is also printed in square brackets ([]) after the command or function name.

For a non-interactive shell, an error condition encountered by a special built-in or other type of utility will cause the shell to write a diagnostic message to standard error and exit as shown in the following table:

Error	Special Built-in	Other Utilities
Shell language syntax error	will exit	will exit
Utility syntax error (option or operand error)	will exit	will not exit
Redirection error	will exit	will not exit
Variable assignment error	will exit	will not exit
Expansion error	will exit	will exit
Command not found	n/a	may exit
Dot script not found	will exit	n/a

An expansion error is one that occurs when the shell expansions are carried out (for example, ${x!y}, because ! is not a valid operator); an implementation may treat these as syntax errors if it is able to detect them during tokenization, rather than during expansion.

If any of the errors shown as "will (may) exit" occur in a subshell, the subshell will (may) exit with a non-zero status, but the script containing the subshell will not exit because of the error.

In all of the cases shown in the table, an interactive shell will write a diagnostic message to standard error without exiting.

USAGE

See largefile(5) for the description of the behavior of ksh and rksh when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXIT STATUS

Each command has an exit status that can influence the behavior of other shell commands. The exit status of commands that are not utilities is documented in this section. The exit status of the standard utilities is documented in their respective sections.

If a command is not found, the exit status will be 127. If the command name is found, but it is not an executable utility, the exit status will be 126. Applications that invoke utilities without using the shell should use these exit status values to report similar errors.

If a command fails during word expansion or redirection, its exit status will be greater than zero.

When reporting the exit status with the special parameter ?, the shell will report the full eight bits of exit status available. The exit status of a command that terminated because it received a signal will be reported as greater than 128.

FILES

/etc/profile
/etc/suid_profile
$HOME/.profile
/tmp/sh*
/dev/null

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/ksh
/usr/bin/rksh

ATTRIBUTE TYPE	ATTRIBUTE VALUE

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled

/usr/xpg4/bin/ksh

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	Enabled

SEE ALSO

cat(1), cd(1), chmod(1), cut(1), echo(1), env(1), getoptcvt(1), jobs(1), login(1), newgrp(1), paste(1), ps(1), shell_builtins(1), stty(1), test(1), vi(1), dup(2), exec(2), fork(2), ioctl(2), lseek(2), pipe(2), ulimit(2), umask(2), wait(2), rand(3C), signal(3C), a.out(4), profile(4), attributes(5), environ(5), largefile(5), signal(5), xpg4(5)

Morris I. Bolsky and David G. Korn, The KornShell Command and Programming Language, Prentice Hall, 1989.

WARNINGS

The use of setuid shell scripts is strongly discouraged.

NOTES

If a command which is a tracked alias is executed, and then a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to exec the original command. Use the -t option of the alias command to correct this situation.

Some very old shell scripts contain a ^ as a synonym for the pipe character |.

Using the fc built-in command within a compound command will cause the whole command to disappear from the history file.

The built-in command . file reads the whole file before any commands are executed. Therefore, alias and unalias commands in the file will not apply to any functions defined in the file.

When the shell executes a shell script that attempts to execute a non-existent command interpreter, the shell returns an erroneous diagnostic message that the shell script file does not exist.

ln

ln - make hard or symbolic links to files

SYNOPSIS

```
/usr/bin/ln [-fns] source_file [target]
/usr/bin/ln [-fns] source_file... target

/usr/xpg4/bin/ln [-fs] source_file [target]
/usr/xpg/bin/ln [-fs] source_file... target
```

DESCRIPTION

In the first synopsis form, the ln utility creates a new directory entry (link) for the file specified by source_file, at the destination path specified by target. If target is not specified, the link is made in the current directory. This first synopsis form is assumed when the final operand does not name an existing directory; if more than two operands are specified and the final is not an existing directory, an error will result.

In the second synopsis form, the ln utility creates a new directory entry for each file specified by a source_file operand, at a destination path in the existing directory named by target.

The ln utility may be used to create both hard links and symbolic links. A hard link is a pointer to a file and 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.

ln by default creates hard links. source_file is linked to target. If target is a directory, another file named source_file is created in target and linked to the original source_file.

/usr/bin/ln

If target is a file, its contents are overwritten. If /usr/bin/ln determines that the mode of target forbids writing, it will print the mode (see chmod(1)), ask for a response, and read the standard input for one line. If the response is affirmative, the link occurs, if permissible;

otherwise, the command exits.

/usr/xpg4/bin/ln
If target is a file and the -f option is not specified, /usr/xpg4/bin/ln will write a diagnostic message to standard error, do nothing more with the current source_file, and go on to any remaining source_files.
A symbolic link is an indirect pointer to a file; its directory entry contains the name of the file to which it is linked. Symbolic links may span file systems and may refer to directories.

File permissions for target may be different from those displayed with a -l listing of the ls(1) command. To display the permissions of target use ls -lL. See stat(2) for more information.

OPTIONS

The following options are supported for both /usr/bin/ln and /usr/xpg4/bin/ln:

-f Link files without questioning the user, even if the mode of target forbids writing. This is the default if the standard input is not a terminal.

-s Create a symbolic link.

If the -s option is used with two arguments, target may be an existing directory or a non-existent file. If target already exists and is not a directory, an error is returned. source_file may be any path name and need not exist. If it exists, it may be a file or directory and may reside on a different file system from target. If target is an existing directory, a file is created in directory target whose name is source_file or the last component of source_file. This file is a symbolic link that references source_file. If target
does not exist, a file with name target is created and it is a symbolic link that references source_file.

If the -s option is used with more than two arguments, target must be an existing directory or an error will be returned. For each source_file, a

link is created in target whose name is the last component of source_file; each new source_file is a symbolic link to the original source_file. The files and target may reside on different file systems.

/usr/bin/ln

The following options are supported for /usr/bin/ln only:

-n If the link is an existing file, do not overwrite the contents of the file. The -f option overrides this option. This is the default behavior for /usr/xpg4/bin/ln, and is silently ignored.

OPERANDS

The following operands are supported:

source_file A path name of a file to be linked. This can be either a regular or special file. If the -s option is specified, source_file can also be a directory.

target The path name of the new directory entry to be created, or of an existing directory in which the new directory entries are to be created.

USAGE

See largefile(5) for the description of the behavior of ln when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of ln: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 All the specified files were linked successfully

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/ln

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled

/usr/xpg4/bin/ln

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	Enabled

SEE ALSO

chmod(1), ls(1), stat(2), attributes(5), environ(5), largefile(5), xpg4(5)

NOTES

A symbolic link to a directory behaves differently than you might expect in certain cases. While an ls(1) on such a link displays the files in the pointed-to directory, an `ls -l' displays information about the link itself:

```
example% ln -s dir link
example% ls link
file1 file2 file3 file4
example% ls -l link
lrwxrwxrwx  1 user           7 Jan 11 23:27 link -> dir
```

When you cd(1) to a directory through a symbolic link, you wind up in the pointed-to location within the file system. This means that the parent of the new working directory is not the parent of the symbolic link, but rather, the parent of the pointed-to directory. For instance, in the following case the final working directory is /usr and not /home/user/linktest.

```
example% pwd
/home/user/linktest
example% ln -s /usr/tmp symlink
example% cd symlink
example% cd ..
```

```
example% pwd
/usr
```

C shell user's can avoid any resulting navigation problems by using the pushd and popd built-in commands instead of cd.

ls

ls - list contents of directory

SYNOPSIS

```
/usr/bin/ls [ -aAbcCdfFgilLmnopqrRstux1 ] [ file...  ]
/usr/xpg4/bin/ls [ -aAbcCdfFgilLmnopqrRstux1 ] [ file...  ]
```

DESCRIPTION

For each file that is a directory, ls lists the contents of the directory; for each file that is an ordinary file, ls repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format for output directed to a terminal is multi-column with entries sorted down the columns. The -1 option allows single column output and -m enables stream output format. In order to determine output formats for the -C, -x, and -m options, ls uses an environment variable, COLUMNS, to determine the number of character positions available on one output line. If this variable is not set, the terminfo(4) database is used to determine the number of columns, based on the environment variable TERM. If this information cannot be obtained, 80 columns are assumed.

The mode printed under the -l option consists of ten characters. The first character may be one of the following:

```
d   the entry is a directory;
D   the entry is a door;
l   the entry is a symbolic link;
b   the entry is a block special file;
c   the entry is a character special file;
p   the entry is a fifo (or "named pipe") special file;
s   the entry is an AF_UNIX address family socket;
-   the entry is an ordinary file;
```

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions;

the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, ``execute'' permission is interpreted to mean permission to search the directory for a specified file. The character after permissions is ACL indication. A plus sign is displayed if there is an ACL associated with the file. Nothing is displayed if there are just permissions. ls -l (the long list) prints its output as follows for the POSIX locale:

```
-rwxrwxrwx+  1 smith  dev    10876  May 16 9:42 part2
```

Reading from right to left, you see that the current directory holds one file, named part2. Next, the last time that file's contents were modified was 9:42 A.M. on May 16. The file contains 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group dev (perhaps indicating ``development''), and his or her login name is smith. The number, in this case 1, indicates the number of links to file part2; see cp(1). The plus sign indicates that there is an ACL associated with the file. Finally, the dash and letters tell you that user, group, and others have permissions to read, write, and execute part2.

The execute (x) symbol here occupies the third position of the three-character sequence. A - in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

- r the file is readable
- w the file is writable
- x the file is executable
- - the indicated permission is not granted
- s the set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on
- S undefined bit-state (the set-user-ID bit is on and the user execution bit is off)
- t the 1000 (octal) bit, or sticky bit, is on (see chmod(1)), and execution is on
- T the 1000 bit is turned on, and execution is off (undefined bit-state)

- l mandatory locking occurs during access (the set-

group-ID bit is on and the group execution bit is off)

/usr/xpg4/bin/ls

L mandatory locking occurs during access (the set-group-ID bit is on and the group execution bit is off)

For user and group permissions, the third position is sometimes occupied by a character other than x or -. s also may occupy this position, referring to the state of the set-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as root but need to assume the identity of the user you login as.
In the case of the sequence of group permissions, l may occupy the third position. l refers to mandatory file and record locking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For others permissions, the third position may be occupied by t or T. These refer to the state of the sticky bit and execution permissions.

OPTIONS

The following options are supported:

-a List all entries, including those that begin with a dot (.), which are normally not listed.

-A List all entries, including those that begin with a dot (.), with the exception of the working directory (.) and the parent directory (..).

-b Force printing of non-printable characters to be in the octal \ddd notation.

-c Use time of last modification of the i-node (file created, mode changed, and so forth) for sorting (-t) or printing (-l or -n).

-C Multi-column output with entries sorted down the columns. This is the default output format.

-d If an argument is a directory, list only its name (not its contents); often used with -l to get the status of

a directory.

-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 Mark directories with a trailing slash (/), doors with a trailing greater-than sign (>), executable files with a trailing asterisk (*), FIFOs with a trailing vertical bar (|), symbolic links with a trailing at-sign (@), and AF_UNIX address family sockets with a trailing equals sign (=).

-g The same as -l, except that the owner is not printed.

-i For each file, print the i-node number in the first column of the report.

-l List in long format, giving mode, ACL indication, number of links, owner, group, size in bytes, and time of last modification for each file (see above). If the file is a special file, the size field instead contains the major and minor device numbers. If the time of last modification is greater than six months ago, it is shown in the format `month date year' for the POSIX locale. When the LC_TIME locale category is not set to the POSIX locale, a different format of the time field may be used. Files modified within six months show `month date time'. If the file is a symbolic link, the filename is printed followed by "->" and the path name of the referenced file.

-L If an argument is a symbolic link, list the file or directory the link references rather than the link itself.

-m Stream output format; files are listed across the page, separated by commas.

-n The same as -l, except that the owner's UID and group's GID numbers are printed, rather than the associated character strings.

-o The same as -l, except that the group is not printed.

-p Put a slash (/) after each filename if the file is a

directory.

-q Force printing of non-printable characters in file names as the character question mark (?).

-r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

-R Recursively list subdirectories encountered.

-s Give size in blocks, including indirect blocks, for each entry.

-t Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See -u and -c.)

-u Use time of last access instead of last modification for sorting (with the -t option) or printing (with the -l option).

-x Multi-column output with entries sorted across rather than down the page.

-1 Print one entry per line of output.

Specifying more than one of the options in the following mutually exclusive pairs is not considered an error: -C and -1 (one), -c and -u. The last option specified in each pair determines the output format.

/usr/bin/ls

Specifying more than one of the options in the following mutually exclusive pairs is not considered an error: -C and -l (ell), -m and -l (ell), -x and -l (ell). The -l option overrides the other option specified in each pair.

/usr/xpg4/bin/ls

Specifying more than one of the options in the following mutually exclusive pairs is not considered an error: -C and -l (ell), -m and -l (ell), -x and -l (ell). The last option specified in each pair determines the output format.

OPERANDS

The following operand is supported:

file A path name of a file to be written. If the file specified is not found, a diagnostic

message will be output on standard error.

USAGE

See largefile(5) for the description of the behavior of ls when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

An example of a file's permissions is:

```
-rwxr--r--
```

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

Another example of a file's permissions is:

```
-rwsr-xr-x
```

This describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

Another example of a file's permissions is:

```
-rw-rwl---
```

This describes a file that is readable and writable only by the user and the group and can be locked during access.

An example of a command line:

```
example% ls -a
```

This command prints the names of all files in the current directory, including those that begin with a dot (.), which normally do not print.

Another example of a command line:

```
example% ls -aisn
```

This command provides information on all files, including those that begin with a dot (a), the i-number-the memory address of the i-node associated with the file-printed in

the left-hand column (i); the size (in blocks) of the files, printed in the column to the right of the i-numbers (s); finally, the report is displayed in the numeric version of the long list, printing the UID (instead of user name) and GID (instead of group name) numbers associated with the files.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of ls: LC_COLLATE, LC_CTYPE, LC_TIME, LC_MESSAGES, NLSPATH, and TZ.

COLUMNS Determine the user's preferred column position width for writing multiple text-column output. If this variable contains a string representing a decimal integer, the ls utility calculates how many path name text columns to write (see -C) based on the width provided. If COLUMNS is not set or invalid, 80 is used. The column width chosen to write the names of files in any given directory will be constant. File names will not be truncated to fit into the multiple text-column output.

EXIT STATUS

0 All information was written successfully.

>0 An error occurred.

FILES

/etc/group	group IDs for ls -l and ls -g
/etc/passwd	user IDs for ls -l and ls -o
/usr/share/lib/terminfo/?/*	terminal information database

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/ls

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled

/usr/xpg4/bin/ls

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	Enabled

SEE ALSO

chmod(1), cp(1), setfacl(1), terminfo(4), attributes(5), environ(5), largefile(5), xpg4(5)

NOTES

Unprintable characters in file names may confuse the columnar output options.

The total block count will be incorrect if if there are hard links among the files.

make

make - maintain, update, and regenerate related programs and files

SYNOPSIS

```
/usr/ccs/bin/make [ -d ] [ -dd ] [ -D ] [ -DD ] [ -e ]
     [ -i ] [ -k ] [ -n ] [ -p ] [ -P ] [ -q ] [ -r ]
     [ -s ] [ -S ] [ -t ] [ -V ] [ -f makefile ] ...
     [ -K statefile ] ...  [ target ] [ macro=value ]

/usr/xpg4/bin/make [ -d ] [ -dd ] [ -D ] [ -DD ] [ -e ]
     [ -i ] [ -k ] [ -n ] [ -p ] [ -P ] [ -q ] [ -r ]
     [ -s ] [ -S ] [ -t ] [ -V ] [ -f makefile ] ...
     [ target ] [ macro=value ]
```

DESCRIPTION

The make utility executes a list of shell commands associated with each target, typically to create or update a file of the same name. makefile contains entries that describe how to bring a target up to date with respect to those on which it depends, which are called dependencies. Since each dependency is a target, it may have dependencies of its own. Targets, dependencies, and sub-dependencies comprise a tree structure that make traces when deciding whether or not to rebuild a target.

The make utility recursively checks each target against its dependencies, beginning with the first target entry in makefile if no target argument is supplied on the command line. If, after processing all of its dependencies, a target file is found either to be missing, or to be older than any of its dependencies, make rebuilds it. Optionally with this version of make, a target can be treated as out-of-date when the commands used to generate it have changed since the last time the target was built.

To build a given target, make executes the list of commands, called a rule. This rule may be listed explicitly in the target's makefile entry, or it may be supplied implicitly by make.

If no target is specified on the command line, make uses the first target defined in makefile.

If a target has no makefile entry, or if its entry has no rule, make attempts to derive a rule by each of the following methods, in turn, until a suitable rule is found. Each method is described under USAGE below.

+ Pattern matching rules.

+ Implicit rules, read in from a user-supplied makefile.

+ Standard implicit rules (also known as suffix rules), typically read in from the file /usr/share/lib/make/make.rules.

+ SCCS retrieval. make retrieves the most recent version from the SCCS history file (if any). See the description of the .SCCS_GET: special-function target for details.

+ The rule from the .DEFAULT: target entry, if there is such an entry in the makefile.

If there is no makefile entry for a target, if no rule can be derived for building it, and if no file by that name is present, make issues an error message and halts.

OPTIONS

The following options are supported:

-d Display the reasons why make chooses to rebuild a target; make displays any and all dependencies that are newer. In addition, make displays options read in from the MAKEFLAGS environment variable.

-dd Display the dependency check and processing in vast detail.

-D Display the text of the makefiles read in.

-DD Display the text of the makefiles, make.rules file, the state file, and all hidden-dependency reports.

-e Environment variables override assignments

within makefiles.

-f makefile Use the description file makefile. A `-' as the makefile argument denotes the standard input. The contents of makefile, when present, override the standard set of implicit rules and predefined macros. When more than one `-f makefile' argument pair appears, make uses the concatenation of those files, in order of appearance.

When no makefile is specified, /usr/ccs/bin/make tries the following in sequence, except when in POSIX mode (see the .POSIX Special-Function Target in the usage section below):

+ If there is a file named makefile in the working directory, make uses that file. If, however, there is an SCCS history file (SCCS/s.makefile) which is newer, make attempts to retrieve and use the most recent version.
+ In the absence of the above file(s), if a file named Makefile is present in the working directory, make attempts to use it. If there is an SCCS history file (SCCS/s.Makefile) that is newer, make attempts to retrieve and use the most recent version.

When no makefile is specified, /usr/ccs/bin/make in POSIX mode and /usr/xpg4/bin/make try the following files in sequence:

+ ./makefile, ./Makefile
+ s.makefile, SCCS/s.makefile
+ s.Makefile, SCCS/s.Makefile

-i Ignore error codes returned by commands. Equivalent to the special-function target `.IGNORE:'.

-k When a nonzero error status is returned by a rule, or when make cannot find a rule, abandon work on the current target, but continue with other dependency branches that do not depend on it.

-K statefile Use the state file statefile. A `-' as the statefile argument denotes the standard input. The contents of statefile, when present, override the standard set of implicit rules and predefined macros. When more than one `-K statefile' argument pair appears, make uses the concatenation of those files, in order of appearance. (see also .KEEP_STATE and .KEEP_STATE_FILE in the Special-Functions Targets section).

-n No execution mode. Print commands, but do not execute them. Even lines beginning with an @ are printed. However, if a command line contains a reference to the $(MAKE) macro, that line is always executed (see the discussion of MAKEFLAGS in Reading Makefiles and the Environment). When in POSIX mode, lines beginning with a "+" are executed.

-p Print out the complete set of macro definitions and target descriptions.

-P Merely report dependencies, rather than building them.

-q Question mode. make returns a zero or nonzero status code depending on whether or not the target file is up to date. When in POSIX mode, lines beginning with a "+" are executed.

-r Do not read in the default makefile /usr/share/lib/make/make.rules.

-s Silent mode. Do not print command lines before executing them. Equivalent to the special-function target .SILENT:.

-S Undo the effect of the -k option. Stop processing when a non-zero exit status is returned by a command.

-t Touch the target files (bringing them up to date) rather than performing their rules. This can be dangerous when files are maintained by more than one person. When the

.KEEP_STATE: target appears in the makefile, this option updates the state file just as if the rules had been performed. When in POSIX mode, lines beginning with a "+" are executed.

-V Puts make into SysV mode. Refer to sysV-make(1) for respective details.

OPERANDS

The following operands are supported:

target Target names, as defined in USAGE.

macro=value

Macro definition. This definition overrides any regular definition for the specified macro within the makefile itself, or in the environment. However, this definition can still be overridden by conditional macro assignments.

USAGE

Refer to make in Programming Utilities Guide for tutorial information.

Reading Makefiles and the environment

When make first starts, it reads the MAKEFLAGS ENVIRONMENT variable to obtain any of the following options specified present in its value: -d, -D, -e, -i, -k, -n, -p, -q, -r, -s, -S, or -t. Due to the implementation of POSIX.2 (see POSIX.2(5), the MAKEFLAGS values will contain a leading `-' character. The make utility then reads the command line for additional options, which also take effect.

Next, make reads in a default makefile that typically contains predefined macro definitions, target entries for implicit rules, and additional rules, such as the rule for retrieving SCCS files. If present, make uses the file make.rules in the current directory; otherwise it reads the file /usr/share/lib/make/make.rules, which contains the standard definitions and rules.

Use the directive:

```
include /usr/share/lib/make/make.rules
```

in your local make.rules file to include them.

Next, make imports variables from the environment (unless the -e option is in effect), and treats them as defined macros. Because make uses the most recent definition it encounters, a macro definition in the makefile normally overrides an environment variable of the same name. When -e is in effect, however, environment variables are read in after all makefiles have been read. In that case, the environment variables take precedence over definitions in the makefile.

Next, make reads any makefiles you specify with -f, or one of makefile or Makefile as described above and then the state file, in the local directory if it exists. If the makefile contains a .KEEP_STATE_FILE target, then it reads the state file that follows the target. Refer to special target .KEEP_STATE_FILE for details.

Next, (after reading the environment if -e is in effect), make reads in any macro definitions supplied as command line arguments. These override macro definitions in the makefile and the environment both, but only for the make command itself.

make exports environment variables, using the most recently defined value. Macro definitions supplied on the command line are not normally exported, unless the macro is also an environment variable.

make does not export macros defined in the makefile. If an environment variable is set, and a macro with the same name is defined on the command line, make exports its value as defined on the command line. Unless -e is in effect, macro definitions within the makefile take precedence over those imported from the environment.

The macros MAKEFLAGS, MAKE, SHELL, HOST_ARCH, HOST_MACH, and TARGET_MACH are special cases. See Special-Purpose Macros, below for details.

Makefile Target Entries

A target entry has the following format:

```
target...  [:|::]  [dependency] ...  [; command] ...
     [command]
```

The first line contains the name of a target, or a space-separated list of target names, terminated with a colon or

double colon. If a list of targets is given, this is equivalent to having a separate entry of the same form for each target. The colon(s) may be followed by a dependency, or a dependency list. make checks this list before building the target. The dependency list may be terminated with a semicolon (;), which in turn can be followed by a single Bourne shell command. Subsequent lines in the target entry begin with a TAB, and contain Bourne shell commands. These commands comprise the rule for building the target.

Shell commands may be continued across input lines by escaping the NEWLINE with a backslash (\). The continuing line must also start with a TAB.

To rebuild a target, make expands macros, strips off initial TAB characters and either executes the command directly (if it contains no shell metacharacters), or passes each command line to a Bourne shell for execution.

The first line that does not begin with a TAB or '#' begins another target or macro definition.

Special Characters

Global

#
: Start a comment. The comment ends at the next NEWLINE. If the `#' follows the TAB in a command line, that line is passed to the shell (which also treats `#' as the start of a comment).

include filename
: If the word include appears as the first seven letters of a line and is followed by a SPACE or TAB, the string that follows is taken as a filename to interpolate at that line. include files can be nested to a depth of no more than about 16. If filename is a macro reference, it is expanded.

Targets and Dependencies

:
: Target list terminator. Words following the colon are added to the dependency list for the target or targets. If a target is named in more than one colon-terminated target entry, the dependencies for all its entries are added to form that target's complete dependency list.

::
Target terminator for alternate dependencies. When used in place of a `:' the double-colon allows a target to be checked and updated with respect to alternate dependency lists. When the target is out-of-date with respect to dependencies listed in the first alternate, it is built according to the rule for that entry. When out-of-date with respect to dependencies in another alternate, it is built according the rule in that other entry. Implicit rules do not apply to double-colon targets; you must supply a rule for each entry. If no dependencies are specified, the rule is always performed.

target [+ target...] :
Target group. The rule in the target entry builds all the indicated targets as a group. It is normally performed only once per make run, but is checked for command dependencies every time a target in the group is encountered in the dependency scan.

%
Pattern matching wild card metacharacter. Like the `*' shell wild card, `%' matches any string of zero or more characters in a target name or dependency, in the target portion of a conditional macro definition, or within a pattern replacement macro reference. Note that only one `%' can appear in a target, dependency-name, or pattern-replacement macro reference.

./pathname
make ignores the leading `./' characters from targets with names given as pathnames relative to "dot," the working directory.

Macros

=
Macro definition. The word to the left of this character is the macro name; words to the right comprise its value. Leading and trailing white space characters are stripped from the value. A word break following the = is implied.

$
Macro reference. The following character, or

the parenthesized or bracketed string, is interpreted as a macro reference: make expands the reference (including the $) by replacing it with the macro's value.

( )
{ }
: Macro-reference name delimiters. A parenthesized or bracketed word appended to a $ is taken as the name of the macro being referred to. Without the delimiters, make recognizes only the first character as the macro name.

$$
: A reference to the dollar-sign macro, the value of which is the character `$'. Used to pass variable expressions beginning with $ to the shell, to refer to environment variables which are expanded by the shell, or to delay processing of dynamic macros within the dependency list of a target, until that target is actually processed.

\$
: Escaped dollar-sign character. Interpreted as a literal dollar sign within a rule.

+=
: When used in place of `=', appends a string to a macro definition (must be surrounded by white space, unlike `=').

:=
: Conditional macro assignment. When preceded by a list of targets with explicit target entries, the macro definition that follows takes effect when processing only those targets, and their dependencies.

:sh =
: Define the value of a macro to be the output of a command (see Command Substitutions, below).

:sh
: In a macro reference, execute the command stored in the macro, and replace the reference with the output of that command (see Command Substitutions).

Rules

+
: make will always execute the commands preceded by a "+", even when -n is specified.

- make ignores any nonzero error code returned by a command line for which the first non-TAB character is a `-'. This character is not passed to the shell as part of the command line. make normally terminates when a command returns nonzero status, unless the -i or -k options, or the .IGNORE: special-function target is in effect.

@ If the first non-TAB character is a @, make does not print the command line before executing it. This character is not passed to the shell.

? Escape command-dependency checking. Command lines starting with this character are not subject to command dependency checking.

! Force command-dependency checking. Command-dependency checking is applied to command lines for which it would otherwise be suppressed. This checking is normally suppressed for lines that contain references to the `?' dynamic macro (for example, `$?').

When any combination of `+', `-', `@', `?', or `!' appear as the first characters after the TAB, all that are present apply. None are passed to the shell.

Special-Function Targets

When incorporated in a makefile, the following target names perform special-functions:

.DEFAULT: If it has an entry in the makefile, the rule for this target is used to process a target when there is no other entry for it, no rule for building it, and no SCCS history file from which to retrieve a current version. make ignores any dependencies for this target.

.DONE: If defined in the makefile, make processes this target and its dependencies after all other targets are built. This target is also performed when make halts with an error,

unless the .FAILED target is defined.

.FAILED: This target, along with its dependencies, is performed instead of .DONE when defined in the makefile and make halts with an error.

.GET_POSIX: This target contains the rule for retrieving the current version of an SCCS file from its history file in the current working directory. make uses this rule when it is running in POSIX mode.

.IGNORE: Ignore errors. When this target appears in the makefile, make ignores non-zero error codes returned from commands. When used in POSIX mode, .IGNORE could be followed by target names only, for which the errors will be ignored.

.INIT: If defined in the makefile, this target and its dependencies are built before any other targets are processed.

.KEEP_STATE: If this target is in effect, make updates the state file, .make.state, in the current directory. This target also activates command dependencies, and hidden dependency checks. If either the .KEEP_STATE: target appears in the makefile, or the environment variable KEEP_STATE is set ("setenv KEEP_STATE"), make will rebuild everything in order to collect dependency information, even if all the targets were up to date due to previous make runs. See also the environment section. This target has no effect if used in POSIX mode.

.KEEP_STATE_FILE:
This target has no effect if used in POSIX mode. This target implies .KEEP_STATE. If the target is followed by a filename, make uses it as the state file. If the target is followed by a directory name, make looks for a .make.state file in that directory. If the target is not followed by any name, make looks for .make.state file in the current working directory.

.MAKE_VERSION: A target-entry of the form:

.MAKE_VERSION: VERSION-number

enables version checking. If the version of make differs from the version indicated, make issues a warning message.

.NO_PARALLEL: Currently, this target has no effect, it is, however, reserved for future use.

.PARALLEL: Currently of no effect, but reserved for future use.

.POSIX: This target enables POSIX mode.

.PRECIOUS: List of files not to delete. make does not remove any of the files listed as dependencies for this target when interrupted. make normally removes the current target when it receives an interrupt. When used in POSIX mode, if the target is not followed by a list of files, all the file are assumed precious.

.SCCS_GET: This target contains the rule for retrieving the current version of an SCCS file from its history file. To suppress automatic retrieval, add an entry for this target with an empty rule to your makefile.

.SCCS_GET_POSIX:
This target contains the rule for retrieving the current version of an SCCS file from its history file. make uses this rule when it is running in POSIX mode.

.SILENT: Run silently. When this target appears in the makefile, make does not echo commands before executing them. When used in POSIX mode, it could be followed by target names, and only those will be executed silently.

.SUFFIXES: The suffixes list for selecting implicit rules (see The Suffixes List).

.WAIT: Currently of no effect, but reserved for future use.

Clearing Special Targets

In this version of make, you can clear the definition of the following special targets by supplying entries for them with no dependencies and no rule:

.DEFAULT, .SCCS_GET, and .SUFFIXES

Command Dependencies

When the .KEEP_STATE: target is effective, make checks the command for building a target against the state file. If the command has changed since the last make run, make rebuilds the target.

Hidden Dependencies

When the .KEEP_STATE: target is effective, make reads reports from cpp(1) and other compilation processors for any "hidden" files, such as #include files. If the target is out of date with respect to any of these files, make rebuilds it.

Macros

Entries of the form

macro=value

define macros. macro is the name of the macro, and value, which consists of all characters up to a comment character or unescaped NEWLINE, is the value. make strips both leading and trailing white space in accepting the value.

Subsequent references to the macro, of the forms: $(name) or ${name} are replaced by value. The parentheses or brackets can be omitted in a reference to a macro with a single-character name.

Macro references can contain references to other macros, in which case nested references are expanded first.

Suffix Replacement Macro References

Substitutions within macros can be made as follows:

$(name:string1=string2)

where string1 is either a suffix, or a word to be replaced in the macro definition, and string2 is the replacement suffix or word. Words in a macro value are separated by SPACE,

TAB, and escaped NEWLINE characters.

Pattern Replacement Macro References

Pattern matching replacements can also be applied to macros, with a reference of the form:

```
$(name:  op%os= np%ns)
```

where op is the existing (old) prefix and os is the existing (old) suffix, np and ns are the new prefix and new suffix, respectively, and the pattern matched by % (a string of zero or more characters), is carried forward from the value being replaced. For example:

```
PROGRAM=fabricate
DEBUG= $(PROGRAM:%=tmp/%-g)
```

sets the value of DEBUG to tmp/fabricate-g.

Note that pattern replacement macro references cannot be used in the dependency list of a pattern matching rule; the % characters are not evaluated independently. Also, any number of % metacharacters can appear after the equal-sign.

Appending to a Macro

Words can be appended to macro values as follows:

```
macro += word ...
```

Special-Purpose Macros

When the MAKEFLAGS variable is present in the environment, make takes options from it, in combination with options entered on the command line. make retains this combined value as the MAKEFLAGS macro, and exports it automatically to each command or shell it invokes.

Note that flags passed by way of MAKEFLAGS are only displayed when the -d, or -dd options are in effect.

The MAKE macro is another special case. It has the value make by default, and temporarily overrides the -n option for any line in which it is referred to. This allows nested invocations of make written as:

```
$(MAKE) ...
```

to run recursively, with the -n flag in effect for all commands but make. This lets you use `make -n' to test an

entire hierarchy of makefiles.

For compatibility with the 4.2 BSD make, the MFLAGS macro is set from the MAKEFLAGS variable by prepending a `-'. MFLAGS is not exported automatically.

The SHELL macro, when set to a single-word value such as /usr/bin/csh, indicates the name of an alternate shell to use. The default is /bin/sh. Note that make executes commands that contain no shell metacharacters itself. Built-in commands, such as dirs in the C shell, are not recognized unless the command line includes a metacharacter (for instance, a semicolon). This macro is neither imported from, nor exported to the environment, regardless of -e. To be sure it is set properly, you must define this macro within every makefile that requires it.
The following macros are provided for use with cross-compilation:

HOST_ARCH — The machine architecture of the host system. By default, this is the output of the arch(1) command prepended with `-'. Under normal circumstances, this value should never be altered by the user.

HOST_MACH — The machine architecture of the host system. By default, this is the output of the mach(1), prepended with `-'. Under normal circumstances, this value should never be altered by the user.

TARGET_ARCH — The machine architecture of the target system. By default, the output of mach, prepended with `-'.

Dynamic Macros

There are several dynamically maintained macros that are useful as abbreviations within rules. They are shown here as references; if you were to define them, make would simply override the definition.

$* — The basename of the current target, derived as if selected for use with an implicit rule.

$< — The name of a dependency file, derived as if selected for use with an implicit rule.

$@ The name of the current target. This is the only dynamic macro whose value is strictly determined when used in a dependency list. (In which case it takes the form `$$@'.)

$? The list of dependencies that are newer than the target. Command-dependency checking is automatically suppressed for lines that contain this macro, just as if the command had been prefixed with a `?'. See the description of `?', under Makefile Special Tokens, above. You can force this check with the ! command-line prefix.

$% The name of the library member being processed. (See Library Maintenance, below.)

To refer to the $@ dynamic macro within a dependency list, precede the reference with an additional `$' character (as in, `$$@'). Because make assigns $< and $* as it would for implicit rules (according to the suffixes list and the directory contents), they may be unreliable when used within explicit target entries.

These macros can be modified to apply either to the filename part, or the directory part of the strings they stand for, by adding an upper case F or D, respectively (and enclosing the resulting name in parentheses or braces). Thus, `$(@D)' refers to the directory part of the string `$@'; if there is no directory part, `.' is assigned. $(@F) refers to the filename part.

Conditional Macro Definitions

A macro definition of the form:

```
target-list := macro = value
```

indicates that when processing any of the targets listed and their dependencies, macro is to be set to the value supplied. Note that if a conditional macro is referred to in a dependency list, the $ must be delayed (use $$ instead). Also, target-list may contain a % pattern, in which case the macro will be conditionally defined for all targets encountered that match the pattern. A pattern replacement reference can be used within the value.

You can temporarily append to a macro's value with a condi-

tional definition of the form:

```
target-list := macro += value
```

Predefined Macros

make supplies the macros shown in the table that follows for compilers and their options, host architectures, and other commands. Unless these macros are read in as environment variables, their values are not exported by make. If you run make with any of these set in the environment, it is a good idea to add commentary to the makefile to indicate what value each is expected to take. If -r is in effect, make does not read the default makefile (./make.rules or /usr/share/lib/make/make.rules) in which these macro definitions are supplied.

Table of Predefined Macros

Use	Macro	Default Value
Library Archives	AR	ar
	ARFLAGS	rv
Assembler Commands	AS	as
	ASFLAGS	
	COMPILE.s	$(AS) $(ASFLAGS)
	COMPILE.S	$(CC) $(ASFLAGS) $(CPPFLAGS) -c
C Compiler Commands	CC	cc
	CFLAGS	
	CPPFLAGS	
	COMPILE.c	$(CC) $(CFLAGS) $(CPPFLAGS) -c
	LINK.c	$(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS)

C++ Compiler Commands	CCC	CC
	CCFLAGS	CFLAGS
	CPPFLAGS	
	COMPILE.cc	$(CCC) $(CCFLAGS) $(CPPFLAGS) -c
	LINK.cc	$(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS)
	COMPILE.C	$(CCC) $(CCFLAGS) $(CPPFLAGS) -c
	LINK.C	$(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS)
FORTRAN 77 Compiler Commands	FC	f77
	FFLAGS	
	COMPILE.f	$(FC) $(FFLAGS) -c
	LINK.f	$(FC) $(FFLAGS) $(LDFLAGS)
	COMPILE.F	$(FC) $(FFLAGS) $(CPPFLAGS) -c
	LINK.F	$(FC) $(FFLAGS) $(CPPFLAGS) $(LDFLAGS)
FORTRAN 90 Compiler Commands	FC	f90
	F90FLAGS	
	COMPILE.f90	$(F90C) $(F90FLAGS) -c
	LINK.f90	$(F90C)
	COMPILE.ftn	$(F90C) $(F90FLAGS) $(CPPFLAGS) -c
	LINK.ftn	$(F90C) $(F90FLAGS) $(CPPFLAGS) $(LDFLAGS)
Link Editor Command	LD	ld
	LDFLAGS	
lex Command	LEX	lex
	LFLAGS	
	LEX.l	$(LEX) $(LFLAGS) -t

lint Command	LINT LINTFLAGS LINT.c	lint $(LINT) $(LINTFLAGS) $(CPPFLAGS)
Modula 2 Commands	M2C M2FLAGS MODFLAGS DEFFLAGS COMPILE.def COMPILE.mod	m2c $(M2C) $(M2FLAGS) $(DEFFLAGS) $(M2C) $(M2FLAGS) $(MODFLAGS)
Pascal Compiler Commands	PC PFLAGS COMPILE.p LINK.p	pc $(PC) $(PFLAGS) $(CPPFLAGS) -c $(PC) $(PFLAGS) $(CPPFLAGS) $(LDFLAGS)
Ratfor Compilation Commands	RFLAGS COMPILE.r LINK.r	 $(FC) $(FFLAGS) $(RFLAGS) -c $(FC) $(FFLAGS) $(RFLAGS) $(LDFLAGS)
rm Command	RM	rm -f

sccs Command	SCCSFLAGS	
	SCCSGETFLAGS	-s
yacc Command	YACC	yacc
	YFLAGS	
	YACC.y	$(YACC) $(YFLAGS)
Suffixes List	SUFFIXES	.o .c .c~ .cc .cc~ .y .y~ .l .l~
		.s .s~ .sh .sh~ .S .S~ .ln .h .h~
		..f .f~ .F .F~ .mod .mod~ .sym
		.def .def~ .p .p~ .r .r~
		.cps .cps~ .C .C~ .Y .Y~
		.L .L .f90 .f90~ .ftn .ftn~

Implicit Rules

When a target has no entry in the makefile, make attempts to determine its class (if any) and apply the rule for that class. An implicit rule describes how to build any target of a given class, from an associated dependency file. The class of a target can be determined either by a pattern, or by a suffix; the corresponding dependency file (with the same basename) from which such a target might be built. In addition to a predefined set of implicit rules, make allows you to define your own, either by pattern, or by suffix.

Pattern Matching Rules

A target entry of the form:

```
tp%ts:  dp%ds
     rule
```

is a pattern matching rule, in which tp is a target prefix, ts is a target suffix, dp is a dependency prefix, and ds is a dependency suffix (any of which may be null). The `%' stands for a basename of zero or more characters that is matched in the target, and is used to construct the name of a dependency. When make encounters a match in its search for an implicit rule, it uses the rule in that target entry to build the target from the dependency file. Pattern-matching implicit rules typically make use of the $@ and $< dynamic macros as placeholders for the target and dependency names. Other, regular dependencies may occur in the dependency list; however, none of the regular dependencies may contain `%'. An entry of the form:

```
tp%ts: [dependency ...] dp%ds [dependency ...]
     rule
```

is a valid pattern matching rule.

Suffix Rules

When no pattern matching rule applies, make checks the target name to see if it ends with a suffix in the known suffixes list. If so, make checks for any suffix rules, as well as a dependency file with same root and another recognized suffix, from which to build it.

The target entry for a suffix rule takes the form:

```
DsTs:
     rule
```

where Ts is the suffix of the target, Ds is the suffix of the dependency file, and rule is the rule for building a target in the class. Both Ds and Ts must appear in the suffixes list. (A suffix need not begin with a `.' to be recognized.)

A suffix rule with only one suffix describes how to build a target having a null (or no) suffix from a dependency file with the indicated suffix. For instance, the .c rule could be used to build an executable program named file from a C source file named `file.c'. If a target with a null suffix

has an explicit dependency, make omits the search for a suffix rule.

Table of Standard Implicit (Suffix) Rules

Use	Implicit Rule Name	Command Line
Assembly	.s.o	$(COMPILE.s) -o $@ $<
Files	.s.a	$(COMPILE.s) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.s~.o	$(-s1GET) $(-s1GFLAGS) -p $< > $*.s $(-s1COMPILE.s) -o $@ $*.s
	.S.o	$(COMPILE.S) -o $@ $<
	.S.a	$(COMPILE.S) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.S~.o	$(GET) $(GFLAGS) -p $< > $*.S $(COMPILE.S) -o $@ $*.S
	.S~.a	$(GET) $(GFLAGS) -p $< > $*.S $(COMPILE.S) -o $% $*.S $(AR) $(ARFLAGS) $@ $% $(RM) $%

```
      C             .c                      $(LINK.c) -o $@ $<
$(LDLIBS)

_____________________________________________________________________
      Files         .c.ln                   $(LINT.c) $(OUTPUT_OPTION)
-i $<

_____________________________________________________________________
                    .c.o                    $(COMPILE.c)
$(OUTPUT_OPTION) $<

_____________________________________________________________________
                    .c.a                    $(COMPILE.c) -o $% $<
                                            $(AR) $(ARFLAGS) $@ $%
                                            $(RM) $%

_____________________________________________________________________
                    ..c~                    $(GET) $(GFLAGS) -p $< >
$*.c
                                            $(CC) $(CFLAGS) $(LDFLAGS)
-o $@ $*.c

_____________________________________________________________________
                    ..c~.o                  $(GET) $(GFLAGS) -p $< >
$*.c
                                            $(CC) $(CFLAGS) -c  $*.c

_____________________________________________________________________
                    ..c~.ln                 $(GET) $(GFLAGS) -p $< >
$*.c
                                            $(LINT.c) $(OUTPUT_OPTION)
-c $*.c

_____________________________________________________________________
                    ..c~.a                  $(GET) $(GFLAGS) -p $< >
$*.c
                                            $(COMPILE.c) -o $% $*.c
                                            $(AR) $(ARFLAGS) $@ $%
                                            $(RM) $%

_____________________________________________________________________
_______
      C++           .cc                     $(LINK.cc) -o $@ $<
$(LDLIBS)

_____________________________________________________________________
```

Files	.cc.o	$(COMPILE.cc) $(OUTPUT_OPTION) $<
	.cc.a	$(COMPILE.cc) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.cc~	$(GET) $(GFLAGS) -p $< > $*.cc $(LINK.cc) -o $@ $*.cc $(LDLIBS)
	.cc.o	$(COMPILE.cc) $(OUTPUT_OPTION) $<
	.cc~.o	$(GET) $(GFLAGS) -p $< > $*.cc $(COMPILE.cc) $(OUTPUT_OPTION) $*.cc
	.cc.a	$(COMPILE.cc) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.cc~.a	$(GET) $(GFLAGS) -p $< > $*.cc $(COMPILE.cc) -o $% $*.cc $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.C	$(LINK.C) -o $@ $< $(LDLIBS)
	.C~	$(GET) $(GFLAGS) -p $< > $*.C $(LINK.C) -o $@ $*.C $(LDLIBS)

	.C.o	$(COMPILE.C) $(OUTPUT_OPTION) $<
	.C~.o	$(GET) $(GFLAGS) -p $< > $*.C $(COMPILE.C) $(OUTPUT_OPTION) $*.C
	.C.a	$(COMPILE.C) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.C~.a	$(GET) $(GFLAGS) -p $< > $*.C $(COMPILE.C) -o $% $*.C $(AR) $(ARFLAGS) $@ $% $(RM) $%
FORTRAN 77	.f	$(LINK.f) -o $@ $< $(LDLIBS)
Files	.f.o	$(COMPILE.f) $(OUTPUT_OPTION) $<
	.f.a	$(COMPILE.f) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.f	$(LINK.f) -o $@ $< $(LDLIBS)
	.f~	$(GET) $(GFLAGS) -p $< > $*.f $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.f
	.f~.o	$(GET) $(GFLAGS) -p $< > $*.f

		$(FC) $(FFLAGS) -c $*.f
	.f~.a	$(GET) $(GFLAGS) -p $< > $*.f $(COMPILE.f) -o $% $*.f $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.F	$(LINK.F) -o $@ $< $(LDLIBS)
	.F.o	$(COMPILE.F) $(OUTPUT_OPTION) $<
	.F.a	$(COMPILE.F) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.F~	$(GET) $(GFLAGS) -p $< > $*.F $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.F
	.F~.o	$(GET) $(GFLAGS) -p $< > $*.F $(FC) $(FFLAGS) -c $*.F
	.F~.a	$(GET) $(GFLAGS) -p $< > $*.F $(COMPILE.F) -o $% $*.F $(AR) $(ARFLAGS) $@ $% $(RM) $%
FORTRAN 90	.f90	$(LINK.f90) -o $@ $< $(LDLIBS)

```
      Files        .f90~                $(GET) $(GFLAGS) -p $< >
$*.f90
                                        $(LINK.f90) -o $@ $*.f90
$(LDLIBS)

_____________________________________________________________________
                   .f90.o               $(COMPILE.f90)
$(OUTPUT_OPTION) $<

_____________________________________________________________________
                   .f90~.o              $(GET) $(GFLAGS) -p $< >
$*.f90
                                        $(COMPILE.f90)
$(OUTPUT_OPTION) $*.f90

_____________________________________________________________________
                   .f90.a               $(COMPILE.f90) -o $% $<
                                        $(AR) $(ARFLAGS) $@ $%
                                        $(RM) $%

_____________________________________________________________________
                   .f90~.a              $(GET) $(GFLAGS) -p $< >
$*.f90
                                        $(COMPILE.f90) -o $%
$*.f90
                                        $(AR) $(ARFLAGS) $@ $%
                                        $(RM) $%

_____________________________________________________________________
                   .ftn                 $(LINK.ftn) -o $@ $<
$(LDLIBS)

_____________________________________________________________________
                   .ftn~                $(GET) $(GFLAGS) -p $< >
$*.ftn
                                        $(LINK.ftn) -o $@ $*.ftn
$(LDLIBS)

_____________________________________________________________________
                   .ftn.o               $(COMPILE.ftn)
$(OUTPUT_OPTION) $<

_____________________________________________________________________
                   .ftn~.o              $(GET) $(GFLAGS) -p $< >
$*.ftn
                                        $(COMPILE.ftn)
$(OUTPUT_OPTION) $*.ftn
```

	.ftn.a	$(COMPILE.ftn) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.ftn~.a	$(GET) $(GFLAGS) -p $< > $*.ftn $(COMPILE.ftn) -o $% $*.ftn $(AR) $(ARFLAGS) $@ $% $(RM) $%
lex Files	.l	$(RM) $*.c $(LEX.l) $< > $*.c $(LINK.c) -o $@ $*.c $(LDLIBS) $(RM) $*.c
	.l.c	$(RM) $@ $(LEX.l) $< > $@
	.l.ln	$(RM) $*.c $(LEX.l) $< > $*.c $(LINT.c) -o $@ -i $*.c $(RM) $*.c
	.l.o	$(RM) $*.c $(LEX.l) $< > $*.c $(COMPILE.c) -o $@ $*.c $(RM) $*.c
	.l~	$(GET) $(GFLAGS) -p $< > $*.l $(LEX) $(LFLAGS) $*.l $(CC) $(CFLAGS) -c lex.yy.c rm -f lex.yy.c mv lex.yy.c $@

	.l~.c	$(GET) $(GFLAGS) -p $< > $*.l $(LEX) $(LFLAGS) $*.l mv lex.yy.c $@
	.l~.ln	$(GET) $(GFLAGS) -p $< > $*.l $(RM) $*.c $(LEX.l) $*.l > $*.c $(LINT.c) -o $@ -i $*.c $(RM) $*.c
	.l~.o	$(GET) $(GFLAGS) -p $< > $*.l $(LEX) $(LFLAGS) $*.l $(CC) $(CFLAGS) -c lex.yy.c rm -f lex.yy.c mv lex.yy.c $@
Modula 2	.mod	$(COMPILE.mod) -o $@ -e $@ $<
Files	.mod.o	$(COMPILE.mod) -o $@ $<
	.def.sym	$(COMPILE.def) -o $@ $<
	.def~.sym	$(GET) $(GFLAGS) -p $< > $*.def $(COMPILE.def) -o $@ $*.def
	.mod~	$(GET) $(GFLAGS) -p $< > $*.mod $(COMPILE.mod) -o $@ -e $@ $*.mod
	.mod~.o	$(GET) $(GFLAGS) -p $< > $*.mod $(COMPILE.mod) -o $@ $*.mod
	.mod~.a	$(GET) $(GFLAGS) -p $< > $*.mod

```
                                        $(COMPILE.mod) -o $%
$*.mod
                                        $(AR) $(ARFLAGS) $@ $%
                                        $(RM) $%
```

```
        NeWS            .cps.h          cps $*.cps

        Files           .cps~.h         $(GET) $(GFLAGS) -p $< >
$*.cps
                                        $(CPS) $(CPSFLAGS) $*.cps
```

```
        Pascal          .p              $(LINK.p) -o $@ $<
$(LDLIBS)

        Files           .p.o            $(COMPILE.p)
$(OUTPUT_OPTION) $<

                        .p~             $(GET) $(GFLAGS) -p $< >
$*.p
                                        $(LINK.p) -o $@ $*.p
$(LDLIBS)

                        .p~.o           $(GET) $(GFLAGS) -p $< >
$*.p
                                        $(COMPILE.p)
$(OUTPUT_OPTION) $*.p

                        .p~.a           $(GET) $(GFLAGS) -p $< >
$*.p
                                        $(COMPILE.p) -o $% $*.p
                                        $(AR) $(ARFLAGS) $@ $%
                                        $(RM) $%
```

```
        Ratfor          .r              $(LINK.r) -o $@ $<
$(LDLIBS)

        Files           .r.o            $(COMPILE.r)
$(OUTPUT_OPTION) $<

                        .r.a            $(COMPILE.r) -o $% $<
                                        $(AR) $(ARFLAGS) $@ $%
```

```
                                        $(RM) $%

                    .r~                 $(GET) $(GFLAGS) -p $< >
$*.r
                                        $(LINK.r) -o $@ $*.r
$(LDLIBS)

                    .r~.o               $(GET) $(GFLAGS) -p $< >
$*.r
                                        $(COMPILE.r)
$(OUTPUT_OPTION) $*.r

                    .r~.a               $(GET) $(GFLAGS) -p $< >
$*.r
                                        $(COMPILE.r) -o $% $*.r
                                        $(AR) $(ARFLAGS) $@ $%
                                        $(RM) $%
```

```
        SCCS        .SCCS_GET           sccs $(SCCSFLAGS) get
$(SCCSGETFLAGS) $@ -G$@
        Files

                    .SCCS_GET_POSIX     sccs $(SCCSFLAGS) get
$(SCCSGETFLAGS) $@

                    .GET_POSIX          $(GET) $(GFLAGS) s.$@
```

```
        Shell       .sh                 cat $< >$@
        Scripts                         chmod +x $@

                    .sh~                $(GET) $(GFLAGS) -p $< >
$*.sh
                                        cp $*.sh $@
                                        chmod a+x $@
```

```
        yacc        .y                  $(YACC.y) $<
        Files                           $(LINK.c) -o $@ y.tab.c
$(LDLIBS)
                                        $(RM) y.tab.c

                    .y.c                $(YACC.y) $<
```

```
                                        mv y.tab.c $@

                    .y.ln               $(YACC.y) $<
                                        $(LINT.c) -o $@ -i y.tab.c
                                        $(RM) y.tab.c

                    .y.o                $(YACC.y) $<
                                        $(COMPILE.c) -o $@ y.tab.c
                                        $(RM) y.tab.c

                    .y~                 $(GET) $(GFLAGS) -p $< >
$*.y
                                        $(YACC) $(YFLAGS) $*.y
                                        $(COMPILE.c) -o $@ y.tab.c
                                        $(RM) y.tab.c

                    .y~.c               $(GET) $(GFLAGS) -p $< >
$*.y
                                        $(YACC) $(YFLAGS) $*.y
                                        mv y.tab.c $@

                    .y~.ln              $(GET) $(GFLAGS) -p $< >
$*.y
                                        $(YACC.y) $*.y
                                        $(LINT.c) -o $@ -i y.tab.c
                |                   |   $(RM) y.tab.c
                |                   |
                |   .y~.o           |   $(GET) $(GFLAGS) -p $< >
$*.y
                |                   |   $(YACC) $(YFLAGS) $*.y
                |                   |   $(CC) $(CFLAGS) -c y.tab.c
                |                   |   rm -f y.tab.c
    |           |                   |   mv y.tab.o $@
|

|___________|___________________|_____________________________________
________|
```

make reads in the standard set of implicit rules from the file /usr/share/lib/make/make.rules, unless -r is in effect, or there is a make.rules file in the local directory that does not include that file.

The Suffixes List

The suffixes list is given as the list of dependencies for the `.SUFFIXES:' special-function target. The default list is contained in the SUFFIXES macro (See Table of Predefined

Macros for the standard list of suffixes). You can define additional .SUFFIXES: targets; a .SUFFIXES target with no dependencies clears the list of suffixes. Order is significant within the list; make selects a rule that corresponds to the target's suffix and the first dependency-file suffix found in the list. To place suffixes at the head of the list, clear the list and replace it with the new suffixes, followed by the default list:

```
.SUFFIXES:
.SUFFIXES: suffixes $(SUFFIXES)
```

A tilde (~) indicates that if a dependency file with the indicated suffix (minus the ~) is under SCCS its most recent version should be retrieved, if necessary, before the target is processed.

Library Maintenance

A target name of the form:

```
lib(member ...)
```

refers to a member, or a space-separated list of members, in an ar(1) library.

The dependency of the library member on the corresponding file must be given as an explicit entry in the makefile. This can be handled by a pattern matching rule of the form:

```
lib(%.s): %.s
```

where .s is the suffix of the member; this suffix is typically .o for object libraries.

A target name of the form

```
lib((symbol))
```

refers to the member of a randomized object library that defines the entry point named symbol.

Command Execution

Command lines are executed one at a time, each by its own process or shell. Shell commands, notably cd, are ineffectual across an unescaped NEWLINE in the makefile. A line is printed (after macro expansion) just before being executed. This is suppressed if it starts with a `@', if there is a `.SILENT:' entry in the makefile, or if make is run with

the -s option. Although the -n option specifies printing without execution, lines containing the macro $(MAKE) are executed regardless, and lines containing the @ special character are printed. The -t (touch) option updates the modification date of a file without executing any rules. This can be dangerous when sources are maintained by more than one person.

make invokes the shell with the -e (exit-on-errors) argument. Thus, with semicolon-separated command sequences, execution of the later commands depends on the success of the former. This behavior can be overridden by starting the command line with a `-', or by writing a shell script that returns a non-zero status only as it finds appropriate.

Bourne Shell Constructs

To use the Bourne shell if control structure for branching, use a command line of the form:

```
if expression ; \
then command ; \
     ... ; \
else command ; \
     ... ; \
fi
```

Although composed of several input lines, the escaped NEWLINE characters insure that make treats them all as one (shell) command line.

To use the Bourne shell for control structure for loops, use a command line of the form:

```
for var in list ; \
     do command; \
     ... ; \
done
```

To refer to a shell variable, use a double-dollar-sign ($$). This prevents expansion of the dollar-sign by make.

Command Substitutions

To incorporate the standard output of a shell command in a macro, use a definition of the form:

```
MACRO:sh =command
```

The command is executed only once, standard error output is discarded, and NEWLINE characters are replaced with SPACEs. If the command has a non-zero exit status, make halts with an error.

To capture the output of a shell command in a macro reference, use a reference of the form:

```
$(MACRO:sh)
```

where MACRO is the name of a macro containing a valid Bourne shell command line. In this case, the command is executed whenever the reference is evaluated. As with shell command substitutions, the reference is replaced with the standard output of the command. If the command has a non-zero exit status, make halts with an error.

In contrast to commands in rules, the command is not subject for macro substitution; therefore, a dollar sign ($) need not be replaced with a double dollar sign ($$).

Signals

INT, SIGTERM, and QUIT signals received from the keyboard halt make and remove the target file being processed unless that target is in the dependency list for .PRECIOUS:.

EXAMPLES

This makefile says that pgm depends on two files a.o and b.o, and that they in turn depend on their corresponding source files (a.c and b.c) along with a common file incl.h:

```
pgm: a.o b.o
     $(LINK.c) -o $@ a.o b.o
a.o: incl.h a.c
     cc -c a.c
b.o: incl.h b.c
     cc -c b.c
```

The following makefile uses implicit rules to express the same dependencies:

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

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of make: LC_CTYPE, LC_MESSAGES, and NLSPATH.

KEEP_STATE

This environment variable has the same effect as the .KEEP_STATE: special-function target. It enables command dependencies, hidden dependencies and writing of the state file.

USE_SVR4_MAKE

This environment variable causes make to invoke the generic System V version of make (/usr/ccs/lib/svr4.make). See sysV-make(1).

MAKEFLAGS This variable is interpreted as a character string representing a series of option characters to be used as the default options. The implementation will accept both of the following formats (but need not accept them when intermixed):

1. The characters are option letters without the leading hyphens or blank character separation used on a command line.

2. The characters are formatted in a manner similar to a portion of the make command line: options are preceded by hyphens and blank-character-separated. The macro=name macro definition operands can also be included. The difference between the contents of MAKEFLAGS and the command line is that the contents of the variable will not be subjected to the word expansions (see wordexp(3C)) associated with parsing the command line values.

When the command-line options -f or -p are used, they will take effect regardless of whether they also appear in MAKEFLAGS. If they otherwise appear in MAKEFLAGS, the result is undefined.

The MAKEFLAGS variable will be accessed from the environment before the makefile is read. At that time, all of the options (except -f and -p) and command-line macros not already included in MAKEFLAGS are added to the MAKEFLAGS

macro. The MAKEFLAGS macro will be passed into the environment as an environment variable for all child processes. If the MAKEFLAGS macro is subsequently set by the makefile, it replaces the MAKEFLAGS variable currently found in the environment.

EXIT STATUS

When the -q option is specified, the make utility will exit with one of the following values:

0 Successful completion.

1 The target was not up-to-date.

>1 An error occurred.

When the -q option is not specified, the make utility will exit with one of the following values:

0 successful completion

>0 an error occurred

FILES

makefile	
Makefile	current version(s) of make description file
s.makefile	
s.Makefile	SCCS history files for the above makefile(s) in the current directory
SCCS/s.makefile	
SCCS/s.Makefile	SCCS history files for the above makefile(s)
make.rules	default file for user-defined targets, macros, and implicit rules
/usr/share/lib/make/make.rules	makefile for standard implicit rules and macros (not read if make.rules is)
.make.state	state file in the local directory

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/ccs/bin/make

ATTRIBUTE TYPE	ATTRIBUTE VALUE

Availability	SUNWsprot

/usr/xpg4/bin/make

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4t

SEE ALSO

ar(1), cd(1), lex(1), sh(1), sccs-get(1), sysV-make(1) yacc(1), passwd(4), attributes(5), POSIX.2(5)

Solaris Advanced User's Guide
Programming Utilities Guide

DIAGNOSTICS

Don't know how to make target 'target'
There is no makefile entry for target, and none of make's implicit rules apply (there is no dependency file with a suffix in the suffixes list, or the target's suffix is not in the list).

***** target removed.**
make was interrupted while building target. Rather than leaving a partially-completed version that is newer than its dependencies, make removes the file named target.

***** target not removed.**
make was interrupted while building target and target was not present in the directory.

***** target could not be removed, reason**
make was interrupted while building target, which was not removed for the indicated reason.

Read of include file `file' failed
The makefile indicated in an include directive was not found, or was inaccessible.

Loop detected when expanding macro value `macro'
A reference to the macro being defined was found in the definition.

Could not write state file `file'
: You used the .KEEP_STATE: target, but do not have write permission on the state file.

*** Error code n
: The previous shell command returned a nonzero error code.

*** signal message
: The previous shell command was aborted due to a signal. If `- core dumped' appears after the message, a core file was created.

Conditional macro conflict encountered
: Displayed only when -d is in effect, this message indicates that two or more parallel targets currently being processed depend on a target which is built differently for each by virtue of conditional macros. Since the target cannot simultaneously satisfy both dependency relationships, it is conflicted.

BUGS

Some commands return nonzero status inappropriately; to overcome this difficulty, prefix the offending command line in the rule with a `-'.

Filenames with the characters `=', `:', or `@', do not work.

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

Options supplied by MAKEFLAGS should be reported for nested make commands. Use the -d option to find out what options the nested command picks up from MAKEFLAGS.

This version of make is incompatible in certain respects with previous versions:

+ The -d option output is much briefer in this version. -dd now produces the equivalent voluminous output.

+ make attempts to derive values for the dynamic macros `$*', `$<', and `$?', while processing explicit targets. It uses the same method as for implicit rules; in some cases this can lead either to unexpected values, or to an empty value being assigned. (Actually, this was true for earlier versions as

well, even though the documentation stated otherwise.)

+ make no longer searches for SCCS history "(s.)" files.

+ Suffix replacement in macro references are now applied after the macro is expanded.

There is no guarantee that makefiles created for this version of make will work with earlier versions.

If there is no make.rules file in the current directory, and the file /usr/share/lib/make/make.rules is missing, make stops before processing any targets. To force make to run anyway, create an empty make.rules file in the current directory.

Once a dependency is made, make assumes the dependency file is present for the remainder of the run. If a rule subsequently removes that file and future targets depend on its existence, unexpected errors may result.

When hidden dependency checking is in effect, the $? macro's value includes the names of hidden dependencies. This can lead to improper filename arguments to commands when $? is used in a rule.

Pattern replacement macro references cannot be used in the dependency list of a pattern matching rule.

Unlike previous versions, this version of make strips a leading `./' from the value of the `$@' dynamic macro.

With automatic SCCS retrieval, this version of make does not support tilde suffix rules.

The only dynamic macro whose value is strictly determined when used in a dependency list is $@ (takes the form `$$@').

make invokes the shell with the -e argument. This cannot be inferred from the syntax of the rule alone.

man

man - find and display reference manual pages

SYNOPSIS

```
man [ - ] [ -adFlrt ] [ -M path ] [ -T macro-package ]
     [-s section ] name ...
man [ -M path ] -k keyword ...
man [ -M path ] -f file ...
```

DESCRIPTION

The man command displays information from the reference manuals. It displays complete manual pages that you select by name, or one-line summaries selected either by keyword (-k), or by the name of an associated file (-f). If no manual page is located, man prints an error message.

Source Format

Reference Manual pages are marked up with either nroff(1) or sgml(5) (Standard Generalized Markup Language) tags. The man command recognizes the type of markup and processes the file accordingly. The various source files are kept in separate directories depending on the type of markup.

Location of Manual Pages

The online Reference Manual page directories are conventionally located in /usr/share/man. The nroff sources are located in the /usr/share/man/man* directories. The SGML sources are located in the /usr/share/man/sman* directories. Each directory corresponds to a section of the manual. Since these directories are optionally installed, they may not reside on your host; you may have to mount /usr/share/man from a host on which they do reside. If there are preformatted, up-to-date versions in the corresponding cat* or fmt* directories, man simply displays or prints those versions. If the preformatted version of interest is out of date or missing, man reformats it prior to display and will store the preformatted version if cat* or fmt* is writable. The windex database is not updated. See catman(1M). If directories for the preformatted versions are not provided, man reformats a page whenever it is requested; it uses a temporary file to store the formatted text during display.

If the standard output is not a terminal, or if the `-' flag is given, man pipes its output through cat(1); otherwise, man pipes its output through more(1) to handle paging and underlining on the screen.

OPTIONS

The following options are supported:

-a Show all manual pages matching name within the MANPATH search path. Manual pages are displayed in the order found.

-d Debug. Displays what a section-specifier evaluates to, method used for searching, and paths searched by man.

-f file...
man attempts to locate manual pages related to any of the given files. It strips the leading path name components from each file, and then prints one-line summaries containing the resulting basename or names. This option also uses the windex database.

-F Force man to search all directories specified by MANPATH or the man.cf file, rather than using the windex lookup database. This is useful if the database is not up to date. If the windex database does not exist, this option is assumed.

-k keyword ...
Print out one-line summaries from the windex database (table of contents) that contain any of the given keywords. The windex database is created using catman(1M).

-l List all manual pages found matching name within the search path.

-M path Specify an alternate search path for manual pages. path is a colon-separated list of directories that contain manual page directory subtrees. For example, if path is /usr/share/man:/usr/local/man, man searches for name in the standard location, and then /usr/local/man. When used with the -k or -f options, the -M option must appear first. Each

directory in the path is assumed to contain sub-directories of the form man* or sman*, one for each section. This option overrides the MANPATH environment variable.

-r Reformat the manual page, but do not display it. This replaces the man - -t name combination.

-s section ...
Specify sections of the manual for man to search. The directories searched for name is limited to those specified by section. section can be a digit (perhaps followed by one or more letters), a word (for example: local, new, old, public), or a letter. To specify multiple sections, separate each section with a comma. This option overrides the MANPATH environment variable and the man.cf file. See Search Path below for an explanation of how man conducts its search.

-t man arranges for the specified manual pages to be troffed to a suitable raster output device (see troff(1). If both the - and -t flags are given, man updates the troffed versions of each named name (if necessary), but does not display them.

-T macro-package
Format manual pages using macro-package rather than the standard -man macros defined in /usr/share/lib/tmac/an. See Search Path under USAGE for a complete explanation of the default search path order.

OPERANDS

The following operand is supported:

name A keyword or the name of a standard utility.

USAGE

Manual Page Sections

Entries in the reference manuals are organized into sections. A section name consists of a major section name, typically a single digit, optionally followed by a subsection name, typically one or more letters. An unadorned major section name acts as an abbreviation for the section of the same name along with all of its subsections. Each

section contains descriptions apropos to a particular reference category, with subsections refining these distinctions. See the intro manual pages for an explanation of the classification used in this release.

Search Path

Before searching for a given name, man constructs a list of candidate directories and sections. man searches for name in the directories specified by the MANPATH ENVIRONMENT variable. If this variable is not set, /usr/share/man is searched by default.

Within the manual page directories, man confines its search to the sections specified in the following order:

- sections specified on the command line with the -s option
- sections embedded in the MANPATH environment variable
- sections specified in the man.cf file for each directory specified in the MANPATH environment variable

If none of the above exist, man searches each directory in the manual page path, and displays the first matching manual page found.

The man.cf file has the following format:

```
MANSECTS=section[,section]...
```

Lines beginning with `#' and blank lines are considered comments, and are ignored. Each directory specified in MANPATH can contain a manual page configuration file, specifying the default search order for that directory.

Formatting Manual Pages

Manual pages are marked up in nroff(1) or SGML. Nroff manual pages are processed by nroff(1) or troff(1) with the -man macro package. Please refer to man(5) for information on macro usage. SGML tagged manual pages are processed by an SGML parser and passed to the formatter.

Preprocessing Nroff Manual Pages

When formatting an nroff manual page, man examines the first line to determine whether it requires special processing.

If the first line is a string of the form:

```
'\"  X
```

where X is separated from the `"' by a single SPACE and consists of any combination of characters in the following list, man pipes its input to troff(1) or nroff(1) through the corresponding preprocessors.

```
e    eqn(1), or neqn for nroff
r    refer(1)
t    tbl(1)
v    vgrind(1)
```

If eqn or neqn is invoked, it will automatically read the file /usr/pub/eqnchar (see eqnchar(5)). If nroff(1) is invoked, col(1) is automatically used.

Referring to Other Nroff Manual Pages

If the first line of the nroff manual page is a reference to another manual page entry fitting the pattern:

```
.so man*/ sourcefile
```

man processes the indicated file in place of the current one. The reference must be expressed as a path name relative to the root of the manual page directory subtree.

When the second or any subsequent line starts with .so, man ignores it; troff(1) or nroff(1) processes the request in the usual manner.

Processing SGML Manual Pages

Manual pages are identified as being marked up in SGML by the presence of the string <!DOCTYPE. If the file also contains the string SHADOW_PAGE the file refers to another manual page for the content. The reference is made with a file entity reference to the manual page that contains the text. This is similar to the .so mechanism used in the nroff formatted man pages.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of man: LC_CTYPE, LC_MESSAGES, and NLSPATH.

MANPATH A colon-separated list of directories; each directory can be followed by a comma-separated list of sections. If set, its value overrides /usr/share/man as the default directory search path, and the man.cf file as the default section search path. The -M and -s flags, in turn, override these values.)

PAGER A program to use for interactively delivering man's output to the screen. If not set, `more -s' is used. See more(1).

TCAT The name of the program to use to display troffed manual pages.

TROFF The name of the formatter to use when the -t flag is given. If not set, troff(1) is used.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

FILES

/usr/share/man	root of the standard manual page directory subtree
/usr/share/man/man?/*	unformatted nroff manual entries
/usr/share/man/sman?/*	unformatted SGML manual entries
/usr/share/man/cat?/*	nroffed manual entries
/usr/share/man/fmt?/*	troffed manual entries
/usr/share/man/windex	table of contents and key-word database
/usr/share/lib/tmac/an	standard -man macro package
/usr/share/lib/sgml/locale/C/dtd/*	SGML document type definition files
/usr/share/lib/sgml/locale/C/solbook/*	SGML style sheet and entity definitions directories
/usr/share/lib/pub/eqnchar	standard definitions for eqn and neqn
man.cf	default search order by section

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdoc
CSI	Enabled (see NOTES)

SEE ALSO

apropos(1), cat(1), col(1), eqn(1), more(1), nroff(1), refer(1), tbl(1), troff(1), vgrind(1), whatis(1), catman(1M), attributes(5), environ(5), eqnchar(5), man(5), sgml(5)

NOTES

The -f and -k options use the windex database, which is created by catman(1M).

The man command is CSI-capable. However, some utilities invoked by the man command, namely, troff, eqn, neqn, refer, tbl, and vgrind, are not verified to be CSI-capable. Because of this, the man command with the -t option may not handle non-EUC data. Also, using the man command to display man pages that require special processing through eqn, neqn, refer, tbl, or vgrind may not be CSI-capable.

BUGS

The manual is supposed to be reproducible either on a phototypesetter or on an ASCII terminal. However, on a terminal some information (indicated by font changes, for instance) is lost.

Some dumb terminals cannot process the vertical motions produced by the e (see eqn(1)) preprocessing flag. To prevent garbled output on these terminals, when you use e also use t, to invoke col(1) implicitly. This workaround has the disadvantage of eliminating superscripts and subscripts - even on those terminals that can display them. Control-q will clear a terminal that gets confused by eqn(1) output.

more

more, page - browse or page through a text file

SYNOPSIS

```
/usr/bin/more [ -cdflrsuw ] [ -lines ] [ +linenumber ]
     [ +/pattern ] [ file ... ]

/usr/bin/page [ -cdflrsuw ] [ -lines ] [ +linenumber ]
     [ +/pattern ] [ file ... ]

/usr/xpg4/bin/more [ -cdeisu ] [ -nnumber ] [ -pcommand ]
     [ -ttagstring ] [ file ... ]

/usr/xpg4/bin/more [ -cdeisu ] [ -nnumber ] [ +command ]
     [ -ttagstring ] [ file ... ]
```

DESCRIPTION

The more utility is a filter that displays the contents of a text file on the terminal, one screenful at a time. It normally pauses after each screenful. /usr/bin/more then prints --More-- and /usr/xpg4/bin/more then prints file at the bottom of the screen. If more is reading from a file rather than a pipe, the percentage of characters displayed so far is also shown.

The more utility scrolls up to display one more line in response to a RETURN character; it displays another screenful in response to a SPACE character. Other commands are listed below.

The page utility clears the screen before displaying the next screenful of text; it only provides a one-line overlap between screens.

The more utility sets the terminal to NOECHO mode, so that the output can be continuous. Commands that you type do not normally show up on your terminal, except for the / and ! commands.

The /usr/bin/more utility exits after displaying the last specified file; /usr/xpg4/bin/more prompts for a command at the last line of the last specified file.

If the standard output is not a terminal, more acts just like cat(1), except that a header is printed before each file in a series.

OPTIONS

The following options are supported for both /usr/bin/more and /usr/xpg4/bin/more:

-c Clear before displaying. Redraws the screen instead of scrolling for faster displays. This option is ignored if the terminal does not have the ability to clear to the end of a line.

-d Display error messages rather than ringing the terminal bell if an unrecognized command is used. This is helpful for inexperienced users.

-s Squeeze. Replace multiple blank lines with a single blank line. This is helpful when viewing nroff(1) output on the screen.

/usr/bin/more

The following options are supported for /usr/bin/more only:

-f Do not fold long lines. This is useful when lines contain nonprinting characters or escape sequences, such as those generated when nroff(1) output is piped through ul(1).

-l Do not treat FORMFEED characters (CTRL-L) as page breaks. If -l is not used, more pauses to accept commands after any line containing a ^L character (CTRL-L). Also, if a file begins with a FORMFEED, the screen is cleared before the file is printed.

-r Normally, more ignores control characters that it does not interpret in some way. The -r option causes these to be displayed as ^C where C stands for any such control character.

-u Suppress generation of underlining escape sequences. Normally, more handles underlin-

ing, such as that produced by nroff(1), in a manner appropriate to the terminal. If the terminal can perform underlining or has a stand-out mode, more supplies appropriate escape sequences as called for in the text file.

-w Normally, more exits when it comes to the end of its input. With -w, however, more prompts and waits for any key to be struck before exiting.

-lines Display the indicated number of lines in each screenful, rather than the default (the number of lines in the terminal screen less two).

+linenumber Start up at linenumber.

+/pattern Start up two lines above the line containing the regular expression pattern. Note: Unlike editors, this construct should not end with a `/.' If it does, then the trailing slash is taken as a character in the search pattern.

/usr/xpg4/bin/more

The following options are supported for /usr/xpg4/bin/more only:

-e Exit immediately after writing the last line of the last file in the argument list.

-i Perform pattern matching in searches without regard to case.

-n number Specify the number of lines per screenful. The number argument is a positive decimal integer. The -n option overrides any values obtained from the environment.

-p command
+command For each file examined, initially execute the more command in the command argument. If the command is a positioning command, such as a line number or a regular expression search, set the current position to represent the

final results of the command, without writing any intermediate lines of the file. For example, the two commands:

```
more -p 1000j file
more -p 1000G file
```

are equivalent and start the display with the current position at line 1000, bypassing the lines that j would write and scroll off the screen if it had been issued during the file examination. If the positioning command is unsuccessful, the first line in the file will be the current position.

-t tagstring Write the screenful of the file containing the tag named by the tagstring argument. See the ctags(1) utility.

-u Treat a backspace character as a printable control character, displayed as a ^H (CTRL-H), suppressing backspacing and the special handling that produces underlined or standout-mode text on some terminal types. Also, do not ignore a carriage-return character at the end of a line.

If both the -t tagstring and -p command (or the obsolescent +command) options are given, the -t tagstring is processed first.

USAGE

ENVIRONMENT

more uses the terminal's terminfo(4) entry to determine its display characteristics.

more looks in the environment variable MORE for any preset options. For instance, to page through files using the -c mode by default, set the value of this variable to -c. (Normally, the command sequence to set up this environment variable is placed in the .login or .profile file).

Commands

The commands take effect immediately. It is not necessary to type a carriage return unless the command requires a file, command, tagstring, or pattern. Up to the time when the command character itself is given, the user may type the

line kill character to cancel the numerical argument being formed. In addition, the user may type the erase character to redisplay the `--More--(xx%)' or file message.

In the following commands, i is a numerical argument (1 by default).

iSPACE Display another screenful, or i more lines if i is specified.

iRETURN Display another line, or i more lines, if specified.

ib
i^B (CTRL-B) Skip back i screenfuls and then print a screenful.

id
i^D (CTRL-D) Scroll forward one half screenful or i more lines. If i is specified, the count becomes the default for subsequent d and u commands.

if Skip i screens full and then print a screenful.

h Help. Give a description of all the more commands.

^L (CTRL-L) Refresh.

in Search for the ith occurrence of the last pattern entered.

q
Q Exit from more.

is Skip i lines and then print a screenful.

v Drop into the vi editor at the current line of the current file.

iz Same as SPACE, except that i, if present, becomes the new default number of lines per screenful.

= Display the current line number.

i/pattern Search forward for the ith occurrence of the regular expression pattern. Display the screenful starting two lines before the line that contains

the ith match for the regular expression pattern, or the end of a pipe, whichever comes first. If more is displaying a file and there is no match, its position in the file remains unchanged. Regular expressions can be edited using erase and kill characters. Erasing back past the first column cancels the search command.

!command Invoke a shell to execute command. The characters % and !, when used within command are replaced with the current filename and the previous shell command, respectively. If there is no current filename, % is not expanded. Prepend a backslash to these characters to escape expansion.

:f Display the current filename and line number.

i:n Skip to the ith next filename given in the command line, or to the last filename in the list if i is out of range.

i:p Skip to the ith previous filename given in the command line, or to the first filename if i is out of range. If given while more is positioned within a file, go to the beginning of the file. If more is reading from a pipe, more simply rings the terminal bell.

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

/usr/bin/more

The following commands are available only in /usr/bin/more:

' Single quote. Go to the point from which the last search started. If no search has been performed in the current file, go to the beginning of the file.

. Dot. Repeat the previous command.

^\ Halt a partial display of text. more stops sending output, and displays the usual --More-- prompt. Some output is lost as a result.

/usr/xpg4/bin/more

The following commands are available only in /usr/xpg4/bin/more:

i^F (CTRL-F) Skip i screens full and print a screenful. (Same as if.)

^G (CTRL-G) Display the current line number (same as =).

ig Go to line number i with the default of the first line in the file.

iG Go to line number i with the default of the Last line in the file.

ij Display another line, or i more lines, if specified. (Same as iRETURN.)

ik Scroll backwards one or i lines, if specified.

mletter Mark the current position with the name letter.

N Reverse direction of search.

r Refresh the screen.

R Refresh the screen, discarding any buffered input.

iu
i^U (CTRL-U) Scroll backwards one half a screen of i lines, if specified. If i is specified, the count becomes the new default for subsequent d and u commands.

ZZ Exit from more (same as q).

:e file Examine (display) a new file. If no file is specified, the current file is redisplayed.

:t tagstring
Go to the tag named by the tagstring argument and scroll/rewrite the screen with the tagged line in the current position. See the ctags utility.

'letter Return to the position that was previously marked with the name letter.

'' Return to the position from which the last move of more than a screenful was made. Defaults to the

beginning of the file.

i?[!]pattern
Search backward in the file for the ith line containing the pattern. The ! specifies to search backward for the ith line that does not contain the pattern.

i/!pattern
Search forward in the file for the ith line that does not contain the pattern.

![command]
Invoke a shell or the specified command.

Large File Behavior
See largefile(5) for the description of the behavior of more and page when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of more: LC_COLLATE (/usr/xpg4/bin/more only), LC_CTYPE, LC_MESSAGES, NLSPATH, and TERM.

/usr/xpg4/bin/more
The following environment variables also affect the execution of /usr/xpg4/bin/more:

COLUMNS — Override the system selected horizontal screen size.

EDITOR — Used by the v command to select an editor.

LINES — Override the system selected vertical screen size. The -n option has precedence over LINES in determining the number of lines in a screen.

MORE — A string specifying options as described in the OPTIONS section, above. As in a command line, The options must be separated by blank characters and each option specification must start with a -. Any command line options are processed after those specified in MORE as though the command line were:

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

FILES

/usr/lib/more.help help file for /usr/bin/more and /usr/bin/page only.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/more /usr/bin/page

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Not enabled

/usr/xpg4/bin/more

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	Enabled

SEE ALSO

cat(1), csh(1), ctags(1), man(1), nroff(1), script(1), sh(1), ul(1), environ(4), terminfo(4), attributes(5), environ(5), largefile(5)

/usr/bin/more /usr/bin/page

regcomp(3C)

/usr/xpg4/bin/more

regex(5), xpg4(5)

NOTES

/usr/bin/more

Skipping backwards is too slow on large files.

/usr/xpg4/bin/more
   This utility will not behave correctly if the terminal is not set up properly.

mv

mv - move files

SYNOPSIS

```
/usr/bin/mv [-fi] source target_file
/usr/bin/mv [-fi] source...  target_dir

/usr/xpg4/bin/mv [-fi] source target_file
/usr/xpg4/bin/mv [-fi] source...  target_dir
```

DESCRIPTION

In the first synopsis form, the mv utility moves the file named by the source operand to the destination specified by the target_file. source and target_file may not have the same name. If target_file does not exist, mv creates a file named target_file. If target_file exists, its contents are overwritten. This first synopsis form is assumed when the final operand does not name an existing directory.

In the second synopsis form, mv moves each file named by a source operand to a destination file in the existing directory named by the target_dir operand. The destination path for each source is the concatenation of the target directory, a single slash character (/), and the last path name component of the source. This second form is assumed when the final operand names an existing directory.

If mv determines that the mode of target_file forbids writing, it will print the mode (see chmod(2)), ask for a response, and read the standard input for one line. If the response is affirmative, the mv occurs, if permissible; otherwise, the command exits. Note that the mode displayed may not fully represent the access permission if target is associated with an ACL. When the parent directory of source is writable and has the sticky bit set, one or more of the following conditions must be true:

- the user must own the file
- the user must own the directory
- the file must be writable by the user
- the user must be a privileged user

If source is a file and target_file is a link to another

file with links, the other links remain and target_file becomes a new file.

OPTIONS

-f mv will move the file(s) without prompting even if it is writing over an existing target. Note that this is the default if the standard input is not a terminal.

-i mv will prompt for confirmation whenever the move would overwrite an existing target. An affirmative answer means that the move should proceed. Any other answer prevents mv from overwriting the target.

/usr/bin/mv

Specifying both the -f and the -i options is not considered an error. The -f option will override the -i option.

/usr/xpg4/bin/mv

Specifying both the -f and the -i options is not considered an error. The last option specified will determine the behavior of mv.

OPERANDS

The following operands are supported:

source A path name of a file or directory to be moved.

target_file A new path name for the file or directory being moved.

target_dir A path name of an existing directory into which to move the input files.

USAGE

See largefile(5) for the description of the behavior of mv when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of mv: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 All input files were moved successfully.

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/mv

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled

/usr/xpg4/bin/mv

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	Enabled

SEE ALSO

cp(1), cpio(1), ln(1), rm(1), setfacl(1), chmod(2), attributes(5), environ(5), largefile(5), xpg4(5)

NOTES

If source and target_dir are on different file systems, mv copies the file and deletes the original; any links to other files are lost.

A `--' permits the user to mark explicitly the end of any command line options, allowing mv to recognize filename arguments that begin with a `-'. As an aid to BSD migration, mv will accept `-' as a synonym for `--'. This migration aid may disappear in a future release. If a `--' and a `-' both appear on the same command line, the second will be interpreted as a filename.

nroff

nroff - format documents for display or line-printer

SYNOPSIS

nroff [-ehiq] [-mname] [-nN] [-opagelist] [-raN] [-sN] [-Tname]

DESCRIPTION

nroff formats text in the named files for typewriter-like devices. See also troff(1).

If no file argument is present, nroff reads the standard input. An argument consisting of a `-' is taken to be a file name corresponding to the standard input.

OPTIONS

Options may appear in any order so long as they appear before the files.

-e Produce equally-spaced words in adjusted lines, using full terminal resolution.

-h Use output TAB characters during horizontal spacing to speed output and reduce output character count. TAB settings are assumed to be every 8 nominal character widths.

-i Read the standard input after the input files are exhausted.

-q Invoke the simultaneous input-output mode of the rd(9F) request.

-mname
Prepend the macro file /usr/share/lib/tmac/tmac.name to the input files.

-nN Number first generated page N.

-opagelist
Print only pages whose page numbers appear in the comma-separated list of numbers and ranges. A range

N-M means pages N through M; an initial -N means from the beginning to page N; and a final N- means from N to the end.

-raN Set register a (one-character) to N.

-sN Stop every N pages. nroff will halt prior to every N pages (default N=1) to allow paper loading or changing, and will resume upon receipt of a NEWLINE.

-Tname
Prepare output for a device of the specified name. Known names are:

37	Teletype Corporation Model 37 terminal - this is the default.
lp | tn300	GE Any line printer or terminal without half-line capability.
300	DASI-300.
300-12	DASI-300 - 12-pitch.
300S	DASI-300S.
300S-12	DASI-300S.
382	DASI-382 (fancy DTC 382).
450	DASI-450 (Diablo Hyterm).
450-12	DASI-450 (Diablo Hyterm) - 12-pitch.
832	AJ 832.

EXAMPLES

The following command:

```
example% nroff -s4 -me users.guide
```

formats users.guide using the -me macro package, and stopping every 4 pages.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of nroff: LC_CTYPE, LC_MESSAGES, and NLSPATH.

FILES

/var/tmp/trtmp*	temporary file
/usr/share/lib/tmac/tmac.*	standard macro files
/usr/share/lib/nterm/*	terminal driving tables for nroff
/usr/share/lib/nterm/README	index to terminal description

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdoc
CSI	Enabled

SEE ALSO

checknr(1), col(1), eqn(1), man(1), tbl(1), troff(1), attributes(5), environ(5), me(5), ms(5), term(5), rd(9F)

od

od - octal dump

SYNOPSIS

```
/usr/bin/od [ -bcCDdFfOoSsvXx ] [ -] [ file ]
     [ offset_string ]
/usr/bin/od [ -bcCDdFfOoSsvXx ] [ -A address_base ]
     [ -j skip ] [ -N count ] [ -t type_string ] ...
     [ - ] [ file

/usr/xpg4/bin/od [ -bcCDdFfOoSsvXx ] [ -] [ file ]
     [ offset_string ]
/usr/xpg4/bin/od [ -bcCDdFfOoSsvXx ] [ -A address_base ]
     [ -j skip ] [ -N count ] [ -t type_string
     [ - ] [ file
```

DESCRIPTION

The od command copies sequentially each input file to standard output and transforming the input data according to the output types specified by the -t or -bcCDdFfOoSsvXx options. If no output type is specified, the default output is as if -t o2 had been specified. Multiple types can be specified by using multiple -bcCDdFfOoSstvXx options. Output lines are written for each type specified in the order in which the types are specified. If no file is specified, the standard input is used. The [offset_string] operand is mutually exclusive from the -A, -j, -N, and -t options. For the purposes of this description, the following terms are used:

word	refers to a 16-bit unit, independent of the word size of the machine
long word	refers to a 32-bit unit
double long word	refers to a 64-bit unit.

OPTIONS

The following options are supported:

-A address_base

Specify the input offset base. The address_base option-argument must be a character. The characters d, o and x specify that the offset base will be written in decimal, octal or hexadecimal, respec-

tively. The character n specifies that the offset will not be written. Unless -A n is specified, the output line will be preceded by the input offset, cumulative across input files, of the next byte to be written. In addition, the offset of the byte following the last byte written will be displayed after all the input data has been processed. Without the -A address_base option and the [offset_string] operand, the input offset base is displayed in octal.

-b Interpret bytes in octal. This is equivalent to -t o1.

/usr/bin/od

-c Display single-byte characters. Certain non-graphic characters appear as C-language escapes:

null	\0
backspace	\b
form-feed	\f
new-line	\n
return	\r
tab	\t

others appear as 3-digit octal numbers. For example:

```
echo "hello world" | od -c
0000000   h   e   l   l   o       w   o   r   l   d
\n
0000014
```

/usr/xpg4/bin/od

-c Interpret bytes as single-byte or multibyte characters according to the current setting of the LC_CTYPE locale category. Printable multibyte characters are written in the area corresponding to the first byte of the character; the two character sequence ** is written in the area corresponding to each remaining byte in the character, as an indication that the character is continued. Non-graphic characters appear the same as they would using the -C option.

-C Interpret bytes as single-byte or multibyte characters according to the current setting of the LC_CTYPE locale category. Printable multibyte characters are written in the area corresponding to the

first byte of the character; two character sequence ** are written in the area corresponding to each remaining byte in the character, as an indication that the character is continued. Certain non-graphic characters appear as C escapes:

null	\0
backspace	\b
formfeed	\f
newline	\n
return	\r
tab	\t

Other non-printable characters appear as one three-digit octal number for each byte in the character.

-d Interpret words in unsigned decimal. This is equivalent to -t u2.

-D Interpret long words in unsigned decimal. This is equivalent to -t u4.

-f Interpret long words in floating point. This is equivalent to -t f4.

-F Interpret double long words in extended precision. This is equivalent to -t f8.

-j skip Jump over skip bytes from the beginning of the input. The od command will read or seek past the first skip bytes in the concatenated input files. If the combined input is not at least skip bytes long, the od command will write a diagnostic message to standard error and exit with a non-zero exit status.

By default, the skip option-argument is interpreted as a decimal number. With a leading 0x or 0X, the offset is interpreted as a hexadecimal number; otherwise, with a leading 0, the offset will be interpreted as an octal number. Appending the character b, k or m to offset will cause it to be interpreted as a multiple of 512, 1024 or 1048576 bytes, respectively. If the skip number is hexadecimal, any appended b is considered to be the final hexadecimal digit. The address is displayed starting at 0000000, and its base is not implied by the base of the skip option-argument.

-N count
Format no more than count bytes of input. By default, count is interpreted as a decimal number. With a leading 0x or 0X, count is interpreted as a hexadecimal number; otherwise, with a leading 0, it is interpreted as an octal number. If count bytes of input (after successfully skipping, if -j skip is specified) are not available, it will not be considered an error; the od command will format the input that is available. The base of the address displayed is not implied by the base of the count option-argument.

-o Interpret words in octal. This is equivalent to -t o2.

-O Interpret long words in unsigned octal. This is equivalent to -t o4.

-s Interpret words in signed decimal. This is equivalent to -t d2.

-S Interpret long words in signed decimal. This is equivalent to -t d4.

-t type_string
Specify one or more output types. The type_string option-argument must be a string specifying the types to be used when writing the input data. The string must consist of the type specification characters:

a Named character. Interpret bytes as named characters. Only the least significant seven bits of each byte will be used for this type specification. Bytes with the values listed in the following table will be written using the corresponding names for those characters.

Named Characters in od

Value	Name	Value	Name	Value	Name	Value	Name
\000	nul	\001	soh	\002	stx	\003	etx
\004	eot	\005	enq	\006	ack	\007	bel
\010	bs	\011	ht	\012	lf	\013	vt

\014	ff	\015	cr	\016	so	\017	si
\020	dle	\021	dc1	\022	dc2	\023	dc3
\024	dc4	\025	nak	\026	syn	\027	etb
\030	can	\031	em	\032	sub	\033	esc
\034	fs	\035	gs	\036	rs	\037	us
\040	sp	\177	del				

c Character. Interpret bytes as single-byte or multibyte characters specified by the current setting of the LC_CTYPE locale category. Printable multibyte characters are written in the area corresponding to the first byte of the character; the two character sequence ** is written in the area corresponding to each remaining byte in the character, as an indication that the character is continued. Certain non-graphic characters appear as C escapes: \0, \a, \b, \f, \n, \r, \t, \v. Other non-printable characters appear as one three-digit octal number for each byte in the character.

The type specification characters d, f, o, u and x can be followed by an optional unsigned decimal integer that specifies the number of bytes to be transformed by each instance of the output type.

f Floating point. Can be followed by an optional F, D or L indicating that the conversion should be applied to an item of type float, double or long double, respectively.

d, o, u and x
Signed decimal, octal, unsigned decimal, and hexadecimal, respectively. Can be followed by an optional C, S, I or L indicating that the conversion should be applied to an item of type char, short, int or long, respectively.

Multiple types can be concatenated within the same type_string and multiple -t options can be specified. Output lines are written for each type specified in the order in which the type specification characters are specified.

-v Show all input data (verbose). Without the -v option, all groups of output lines that would be identical to the immediately preceding output line

(except for byte offsets), will be replaced with a line containing only an asterisk (*).

-x Interpret words in hex. This is equivalent to -t x2.

-X Interpret long words in hex. This is equivalent to -t x4.

OPERANDS

The following operands are supported for both /usr/bin/od and /usr/xpg4/bin/od:

- Use the standard input in addition to any files specified. When this operand is not given, the standard input is used only if no file operands are specified.

/usr/bin/od

The following operands are supported for /usr/bin/od only:

file A path name of a file to be read. If no file operands are specified, the standard input will be used. If there are no more than two operands, none of the -A, -j, -N or -t options is specified, and any of the following are true:

1. the first character of the last operand is a plus sign (+)

2. the first character of the second operand is numeric

3. the first character of the second operand is x and the second character of the second operand is a lower-case hexadecimal character or digit

4. the second operand is named "x"

5. the second operand is named

"."
then the corresponding operand is assumed to be an offset operand rather than a file operand.

Without the -N count option, the display continues until an end-of-file is reached.

[+] [0] offset [.][b|B]
[+] [0] [offset] [.]
[+][0x|x][offset]
[+][0x|x] offset[B]

The offset_string operand specifies the byte offset in the file where dumping is to commence. The offset is interpreted in octal bytes by default. If offset begins with "0", it is interpreted in octal. If offset begins with "x" or "0x", it is interpreted in hexadecimal and any appended "b" is considered to be the final hexadecimal digit. If "." is appended, the offset is interpreted in decimal. If "b" or "B" is appended, the offset is interpreted in units of 512 bytes. If the file argument is omitted, the offset argument must be preceded by a plus sign (+). The address is displayed starting at the given offset. The radix of the address will be the same as the radix of the offset, if specified, otherwise it will be octal. Decimal overrides octal, and it is an error to specify both hexadecimal and decimal conversions in the same offset operand.

/usr/xpg4/bin/od

The following operands are supported for /usr/xpg4/bin/od only:

file

Same as /usr/bin/od, except only one of the first two conditions must be true.

[+] [0] offset [.][b|B]
+ [offset] [.]

[+][0x][offset]
[+][0x] offset[B]
+x [offset]
+xoffset [B] Description of offset_string is the same as for /usr/bin/od.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of od: LC_CTYPE, LC_MESSAGES, LC_NUMERIC, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 Successful completion.
>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/od

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtoo
CSI	enabled

/usr/xpg4/bin/od

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	enabled

SEE ALSO

sed(1), attributes(5), environ(5), xpg4(5)

printf

printf - write formatted output

SYNOPSIS

printf format [argument...]

DESCRIPTION

The printf command writes formatted operands to the standard output. The argument operands are formatted under control of the format operand.

OPERANDS

The following operands are supported:

format — A string describing the format to use to write the remaining operands. The format operand is used as the format string described on the formats(5) manual page, with the following exceptions:

+ A SPACE character in the format string, in any context other than a flag of a conversion specification, is treated as an ordinary character that is copied to the output.

+ A delta character in the format string is treated as a delta character, not as a SPACE character.

+ In addition to the escape sequences described on the formats(5) manual page (\\, \a, \b, \f, \n, \r, \t, \v), \ddd, where ddd is a one-, two- or three-digit octal number, is written as a byte with the numeric value specified by the octal number.

+ The program does not precede or follow output from the d or u conversion specifications with blank characters not specified by the format operand.

+ The program does not precede output from the o conversion specification with zeros not specified by the format operand.

+ An additional conversion character, b, is supported as follows. The argument is taken to be a string that may contain backslash-escape sequences. The following backslash-escape sequences are supported:
  - the escape sequences listed on the formats(5) manual page (\\, \a, \b, \f, \n, \r, \t, \v), which are converted to the characters they represent

  - \0ddd, where ddd is a zero-, one-, two- or three-digit octal number that is converted to a byte with the numeric value specified by the octal number

  - \c, which is written and causes printf to ignore any remaining characters in the string operand containing it, any remaining string operands and any additional characters in the format operand.

The interpretation of a backslash followed by any other sequence of characters is unspecified.

Bytes from the converted string are written until the end of the string or the number of bytes indicated by the precision specification is reached. If the precision is omitted, it is taken to be infinite, so all bytes up to the end of the converted string are written. For each specification that consumes an argument, the next argument operand is evaluated and converted to the appropriate type for the conversion as specified below. The format operand is reused as often as necessary to satisfy the argument operands. Any extra c or s conversion specifications are evaluated as if a null string argument were supplied; other extra conversion specifications are evaluated as if a zero argument were supplied. If the format operand contains no conversion specifications and argu-

ment operands are present, the results are unspecified. If a character sequence in the format operand begins with a % character, but does not form a valid conversion specification, the behavior is unspecified.

argument The strings to be written to standard output, under the control of format. The argument operands are treated as strings if the corresponding conversion character is b, c or s; otherwise, it is evaluated as a C constant, as described by the ISO C standard, with the following extensions:

+ A leading plus or minus sign is allowed.

+ If the leading character is a single- or double-quote, the value is the numeric value in the underlying codeset of the character following the single- or double-quote.

If an argument operand cannot be completely converted into an internal value appropriate to the corresponding conversion specification, a diagnostic message is written to standard error and the utility does not exit with a zero exit status, but continues processing any remaining operands and writes the value accumulated at the time the error was detected to standard output.

USAGE

Note that this printf utility, like the printf(3S) function on which it is based, makes no special provision for dealing with multi-byte characters when using the %c conversion specification or when a precision is specified in a %b or %s conversion specification. Applications should be extremely cautious using either of these features when there are multi-byte characters in the character set.

Field widths and precisions cannot be specified as *.

For compatibility with previous versions of SunOS 5.x, the $ format specifier is supported for formats containing only %s specifiers.

The %b conversion specification is not part of the ISO C standard; it has been added here as a portable way to process backslash escapes expanded in string operands as provided by the echo utility. See also the USAGE section of the echo(1) manual page for ways to use printf as a replacement for all of the traditional versions of the echo utility.

If an argument cannot be parsed correctly for the corresponding conversion specification, the printf utility reports an error. Thus, overflow and extraneous characters at the end of an argument being used for a numeric conversion are to be reported as errors.

It is not considered an error if an argument operand is not completely used for a c or s conversion or if a string operand's first or second character is used to get the numeric value of a character.

EXAMPLES

To alert the user and then print and read a series of prompts:

```
printf "\aPlease fill in the following: \nName: "
read name
printf "Phone number: "
read phone
```

To read out a list of right and wrong answers from a file, calculate the percentage correctly, and print them out. The numbers are right-justified and separated by a single tab character. The percentage is written to one decimal place of accuracy:

```
while read right wrong ; do
   percent=$(echo "scale=1;($right*100)/($right+$wrong)"
| bc)
   printf "%2d right\t%2d wrong\t(%s%%)\n" \
        $right $wrong $percent
done < database_file
```

The command:

```
printf "%5d%4d\n" 1 21 321 4321 54321
```

produces:

```
    1  21
```

```
  3214321
54321   0
```

Note that the format operand is used three times to print all of the given strings and that a 0 was supplied by printf to satisfy the last %4d conversion specification.

The printf utility tells the user when conversion errors are detected while producing numeric output; thus, the following results would be expected on an implementation with 32-bit twos-complement integers when %d is specified as the format operand:

Argument	Standard Output	Diagnostic Output
5a	5	printf: 5a not completely converted
9999999999	2147483647	printf: 9999999999: Results too large
-9999999999	-2147483648	printf: -9999999999: Results too large
ABC	0	printf: ABC expected numeric value

Note that the value shown on standard output is what would be expected as the return value from the function strtol(3C). A similar correspondence exists between %u and strtoul(3C), and %e, %f and %g and strtod(3C).
In a locale using the ISO/IEC 646:1991 standard as the underlying codeset, the command:

```
printf "%d\n" 3 +3 -3 \'3 \"+3 "'-3"
```

produces:

```
3    Numeric value of constant 3
3    Numeric value of constant 3
-3   Numeric value of constant -3
51   Numeric value of the character `3' in the ISO/IEC
     646:1991 standard codeset
43   Numeric value of the character `+' in the ISO/IEC
     646:1991 standard codeset
45   Numeric value of the character `-' in the SO/IEC
     646:1991 standard codeset
```

Note that in a locale with multi-byte characters, the value of a character is intended to be the value of the equivalent of the wchar_t representation of the character.

If an argument operand cannot be completely converted into an internal value appropriate to the corresponding conversion specification, a diagnostic message is written to standard error and the utility does exit with a zero exit status, but continues processing any remaining operands and writes the value accumulated at the time the error was detected to standard output.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of printf: LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_TIME, TZ, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWloc
CSI	enabled

SEE ALSO

awk(1), bc(1), echo(1), printf(3S), strtod(3C), strtol(3C), strtoul(3C), attributes(5), environ(5), formats(5)

ps

ps - report process status

SYNOPSIS

```
ps [ -aAcdefjlLPy ] [ -g grplist ] [ -n namelist ]
     [[ -o format ] ... ] [ -p proclist ]
     [ -s sidlist ] [ -t term ] [ -u uidlist ]
     [ -U uidlist ] [ -G gidlist ]
```

DESCRIPTION

The ps command prints information about active processes. Without options, ps prints information about processes associated with the controlling terminal. The output contains only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the options.

Some options accept lists as arguments. Items in a list can be either separated by commas or else enclosed in quotes and separated by commas or spaces. Values for proclist and grplist must be numeric.

OPTIONS

The following options are supported:

-a List information about all processes most frequently requested: all those except process group leaders and processes not associated with a terminal.

-A List information for all processes. Identical to -e, below.

-c Print information in a format that reflects scheduler properties as described in priocntl(1). The -c option affects the output of the -f and -l options, as described below.

-d List information about all processes except session leaders.

-e List information about every process now running.

-f Generate a full listing. (See below for significance of columns in a full listing.)

-g grplist List only process data whose group leader's ID number(s) appears in grplist. (A group leader is a process whose process ID number is identical to its process group ID number.)

-G gidlist List information for processes whose real group ID numbers are given in gidlist. The gidlist must be a single argument in the form of a blank- or comma-separated list.

-j Print session ID and process group ID.

-l Generate a long listing. (See below.)

-L Print information about each light weight process (lwp) in each selected process. (See below.)

-n namelist Specify the name of an alternative system namelist file in place of the default. This option is accepted for compatibility, but is ignored.

-o format Print information according to the format specification given in format. This is fully described in DISPLAY FORMATS. Multiple -o options can be specified; the format specification will be interpreted as the space-character-separated concatenation of all the format option-arguments.

-p proclist List only process data whose process ID numbers are given in proclist.

-P Print the number of the processor to which the process or lwp is bound, if any, under an additional column header, PSR.

-s sidlist List information on all session leaders whose IDs appear in sidlist.

-t term List only process data associated with term.

Terminal identifiers are specified as a device file name, and an identifier. For example, term/a, or pts/0.

-u uidlist List only process data whose effective user ID number or login name is given in uidlist. In the listing, the numerical user ID will be printed unless you give the -f option, which prints the login name.

-U uidlist List information for processes whose real user ID numbers or login names are given in uidlist. The uidlist must be a single argument in the form of a blank- or comma-separated list.

-y Under a long listing (-l), omit the obsolete F and ADDR columns and include an RSS column to report the resident set size of the process. Under the -y option, both RSS and SZ (see below) will be reported in units of kilobytes instead of pages.

Many of the options shown are used to select processes to list. If any are specified, the default list will be ignored and ps will select the processes represented by the inclusive OR of all the selection-criteria options.

DISPLAY FORMATS

Under the -f option, ps tries to determine the command name and arguments given when the process was created by examining the user block. Failing this, the command name is printed, as it would have appeared without the -f option, in square brackets.

The column headings and the meaning of the columns in a ps listing are given below; the letters f and l indicate the option (full or long, respectively) that causes the corresponding heading to appear; all means that the heading always appears. Note: These two options determine only what information is provided for a process; they do not determine which processes will be listed.

F (l) Flags (hexadecimal and additive) associated with the process. These flags are available for historical purposes; no meaning should be currently ascribed to them.

S	(l)	The state of the process: O Process is running on a processor. S Sleeping: process is waiting for an event to complete. R Runnable: process is on run queue. Z Zombie state: process terminated and parent not waiting. T Process is stopped, either by a job control signal or because it is being traced.
UID	(f,l)	The effective user ID number of the process (the login name is printed under the -f option).
PID	(all)	The process ID of the process (this datum is necessary in order to kill a process).
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilization for scheduling (obsolete). Not printed when the -c option is used.
CLS	(f,l)	Scheduling class. Printed only when the -c option is used.
PRI	(l)	The priority of the process. Without the -c option, higher numbers mean lower priority. With the -c option, higher numbers mean higher priority.
NI	(l)	Nice value, used in priority computation. Not printed when the -c option is used. Only processes in the certain scheduling classes have a nice value.
ADDR	(l)	The memory address of the process.
SZ	(l)	The total size of the process in virtual memory, including all mapped files and devices, in pages. See pagesize(1).
WCHAN	(l)	The address of an event for which the process is sleeping (if blank, the pro-

cess is running).

STIME	(f)	The starting time of the process, given in hours, minutes, and seconds. (A process begun more than twenty-four hours before the ps inquiry is executed is given in months and days.)
TTY	(all)	The controlling terminal for the process (the message, ?, is printed when there is no controlling terminal).
TIME	(all)	The cumulative execution time for the process.
CMD	(all)	The command name (the full command name and its arguments, up to a limit of 80 characters, are printed under the -f option).

The following two additional columns are printed when the -j option is specified:

PGID	The process ID of the process group leader.
SID	The process ID of the session leader.

The following two additional columns are printed when the -L option is specified:

LWP	The lwp ID of the lwp being reported.
NLWP	The number of lwps in the process (if -f is also specified).

Under the -L option, one line is printed for each lwp in the process and the time-reporting fields STIME and TIME show the values for the lwp, not the process. A traditional single-threaded process contains only one lwp.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked <defunct>.

-o format
The -o option allows the output format to be specified under user control.

The format specification must be a list of names presented as a single argument, blank- or comma-separated. Each variable has a default header. The default header can be overridden by appending an equals sign and the new text of the header. The rest of the characters in the argument will be used as the header text. The fields specified will be written in the order specified on the command line, and should be arranged in columns in the output. The field widths will be selected by the system to be at least as wide as the header text (default or overridden value). If the header text is null, such as -o user=, the field width will be at least as wide as the default header text. If all header text fields are null, no header line will be written.

The following names are recognized in the POSIX locale:

user The effective user ID of the process. This will be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

ruser The real user ID of the process. This will be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

group The effective group ID of the process. This will be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

rgroup The real group ID of the process. This will be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

pid The decimal value of the process ID.

ppid The decimal value of the parent process ID.

pgid The decimal value of the process group ID.

pcpu The ratio of CPU time used recently to CPU time available in the same period, expressed as a percentage. The meaning of ``recently'' in this context is unspecified. The CPU time available is determined in an unspecified manner.

vsz The total size of the process in virtual memory, in kilobytes.

nice The decimal value of the system scheduling priority of the process. See nice(1).

etime In the POSIX locale, the elapsed time since the process was started, in the form:
[[dd-]hh:]mm:ss

where

dd will represent the number of days,
hh the number of hours,
mm the number of minutes, and
ss the number of seconds.

The dd field will be a decimal integer. The hh, mm and ss fields will be two-digit decimal integers padded on the left with zeros.

time In the POSIX locale, the cumulative CPU time of the process in the form:
[dd-]hh:mm:ss

The dd, hh, mm, and ss fields will be as described in the etime specifier.

tty The name of the controlling terminal of the process (if any) in the same format used by the who(1) command.

comm The name of the command being executed (argv[0] value) as a string.

args The command with all its arguments as a string. The implementation may truncate this value to the field width; it is implementation-dependent whether any further truncation occurs. It is unspecified whether the string represented is a version of the argument list as it was passed to the command when it started, or is a version of the arguments as they may have been modified by the application. Applications cannot depend on being able to modify their argument list and having that modification be reflected in the output of ps. The Solaris implementation limits the string to 80 bytes; the string is the ver-

sion of the argument list as it was passed to the command when it started.

The following names are recognized in the Solaris implementation:

f — Flags (hexadecimal and additive) associated with the process.

s — The state of the process.

c — Processor utilization for scheduling (obsolete).

uid — The effective user ID number of the process as a decimal integer.

ruid — The real user ID number of the process as a decimal integer.

gid — The effective group ID number of the process as a decimal integer.

rgid — The real group ID number of the process as a decimal integer.

sid — The process ID of the session leader.

class — The scheduling class of the process.

pri — The priority of the process. Higher numbers mean higher priority.

opri — The obsolete priority of the process. Lower numbers mean higher priority.

lwp — The decimal value of the lwp ID. Requesting this formatting option causes one line to be printed for each lwp in the process.

nlwp — The number of lwps in the process.

psr — The number of the processor to which the process or lwp is bound.

addr — The memory address of the process.

osz — The total size of the process in virtual memory, in pages.

wchan The address of an event for which the process is sleeping (if -, the process is running).

stime The starting time or date of the process, printed with no blanks.

rss The resident set size of the process, in kilobytes.

pmem The ratio of the process's resident set size to the physical memory on the machine, expressed as a percentage.

fname The first 8 bytes of the base name of the process's executable file.

Only comm and args are allowed to contain blank characters; all others, including the Solaris implementation variables, are not.

The following table specifies the default header to be used in the POSIX locale corresponding to each format specifier.

Format Specifier	Default Header	Format Specifier	Default Header
args	COMMAND	ppid	PPID
comm	COMMAND	rgroup	RGROUP
etime	ELAPSED	ruser	RUSER
group	GROUP	time	TIME
nice	NI	tty	TT
pcpu	%CPU	user	USER
pgid	PGID	vsz	VSZ

pid	PID

The following table lists the Solaris implementation format specifiers and the default header used with each.

Format Specifier	Default Header	Format Specifier	Default Header
addr	ADDR	pri	PRI
c	C	psr	PSR
class	CLS	rgid	RGID
f	F	rss	RSS
fname	COMMAND	ruid	RUID
gid	GID	s	S
lwp	LWP	sid	SID
nlwp	NLWP	stime	STIME
opri	PRI	uid	UID
osz	SZ	wchan	WCHAN
pmem	%MEM		

EXAMPLES

The command:

```
example% ps -o user,pid,ppid=MOM -o args
```

writes the following in the POSIX locale:

```
  USER   PID   MOM   COMMAND
helene    34    12   ps -o uid,pid,ppid=MOM -o args
```

The contents of the COMMAND field need not be the same due to possible truncation.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of ps: LC_CTYPE, LC_MESSAGES, LC_TIME, and NLSPATH.

COLUMNS — Override the system-selected horizontal screen size, used to determine the number of text columns to display.

EXIT STATUS

The following exit values are returned:

0 — Successful completion.

>0 — An error occurred.

FILES

/dev/pts/*	
/dev/term/*	terminal (``tty'') names searcher files
/etc/passwd	UID information supplier
/proc/*	process control files
/tmp/ps_data	internal data structure

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled (see NOTES)

SEE ALSO

kill(1), nice(1), pagesize(1), priocntl(1), who(1), getty(1M), proc(4), ttysrch(4), attributes(5), environ(5)

NOTES

Things can change while ps is running; the snap-shot it gives is true only for a split-second, and it may not be accurate by the time you see it. Some data printed for defunct processes is irrelevant.

If no options to select processes are specified, ps will report all processes associated with the controlling terminal. If there is no controlling terminal, there will be no report other than the header.

ps -ef or ps -o stime may not report the actual start of a tty login session, but rather an earlier time, when a getty was last respawned on the tty line.

ps is CSI-enabled except for login names (usernames).

regexp

regexp, compile, step, advance - simple regular expression compile and match routines

SYNOPSIS

```
#define INIT declarations
#define GETC(void) getc code
#define PEEKC(void) peekc code
#define UNGETC(void) ungetc code
#define RETURN(ptr) return code
#define ERROR(val) error code

#include <regexp.h>

char *compile(char *instring, char *expbuf,
     const char *endbuf, int eof);

int step(const char *string, const char *expbuf);

int advance(const char *string, const char *expbuf);

extern char *loc1, *loc2, *locs;
```

DESCRIPTION

Regular Expressions (REs) provide a mechanism to select specific strings from a set of character strings. The Simple Regular Expressions described below differ from the Internationalized Regular Expressions described on the regex(5) manual page in the following ways:

+ only Basic Regular Expressions are supported

+ the Internationalization features-character class, equivalence class, and multi-character collation-are not supported.

The functions step(), advance(), and compile() are general purpose regular expression matching routines to be used in programs that perform regular expression matching. These functions are defined by the <regexp.h> header.

The functions step() and advance() do pattern matching given a character string and a compiled regular expression as

input.

The function compile() takes as input a regular expression as defined below and produces a compiled expression that can be used with step() or advance().

Basic Regular Expressions

A regular expression specifies a set of character strings. A member of this set of strings is said to be matched by the regular expression. Some characters have special meaning when used in a regular expression; other characters stand for themselves.

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 4.1 and 4.3 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).

c. $ (dollar sign), which is special at the end of an entire RE (see 4.2 below).

d. The character used to bound (that is, 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 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); for example, []a-f] matches either a right square bracket (]) or one of the ASCII 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 REs 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 0 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 consist-

ing of two repeated appearances of the same string.

A RE may be constrained to match words.

3.1 \< constrains a RE to match the beginning of a string or to follow a character that is not a digit, underscore, or letter. The first character matching the RE must be a digit, underscore, or letter.

3.2 \> constrains a RE to match the end of a string or to precede a character that is not a digit, underscore, or letter.

An entire RE may be constrained to match only an initial segment or final segment of a line (or both).

4.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an initial segment of a line.

4.2 A dollar sign ($) at the end of an entire RE constrains that RE to match a final segment of a line.

4.3 The construction ^entire RE$ constrains the entire RE to match the entire line.

The null RE (for example, //) is equivalent to the last RE encountered.

Addressing with REs

Addresses are constructed as follows:

1. The character "." addresses the current line.

2. The character "$" addresses the last line of the buffer.

3. A decimal number n addresses the n-th line of the buffer.

4. 'x addresses the line marked with the mark name character x, which must be an ASCII lower-case letter (a-z). Lines are marked with the k command described below.

5. A RE enclosed by slashes (/) addresses the first line found by searching forward from the line following the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning

of the buffer and continues up to and including the current line, so that the entire buffer is searched.

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.

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. A shorthand for .+5 is .5.

8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; for example, -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,$, while a semicolon (;) stands for the pair .,$.

Characters With Special Meaning

Characters that have special meaning except when they appear within square brackets ([]) or are preceded by \ are: ., *, [, \. Other special characters, such as $ have special meaning in more restricted contexts.

The character ^ at the beginning of an expression permits a successful match only immediately after a newline, and the character $ at the end of an expression requires a trailing newline.

Two characters have special meaning only when used within square brackets. The character - denotes a range, [c-c],

unless it is just after the open bracket or before the closing bracket, [-c] or [c-] in which case it has no special meaning. When used within brackets, the character ^ has the meaning complement of if it immediately follows the open bracket (example: [^c]); elsewhere between brackets (example: [c^]) it stands for the ordinary character ^.

The special meaning of the \ operator can be escaped only by preceding it with another \, for example \\.

Macros

Programs must have the following five macros declared before the #include <regexp.h> statement. These macros are used by the compile() routine. The macros GETC, PEEKC, and UNGETC operate on the regular expression given as input to compile().

GETC — This macro returns the value of the next character (byte) in the regular expression pattern. Successive calls to GETC should return successive characters of the regular expression.

PEEKC — This macro returns the next character (byte) in the regular expression. Immediately successive calls to PEEKC should return the same character, which should also be the next character returned by GETC.

UNGETC — This macro causes the argument c to be returned by the next call to GETC and PEEKC. No more than one character of pushback is ever needed and this character is guaranteed to be the last character read by GETC. The return value of the macro UNGETC(c) is always ignored.

RETURN(ptr) — This macro is used on normal exit of the compile() routine. The value of the argument ptr is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.

ERROR(val) — This macro is the abnormal return from the compile() routine. The argument val is an error number (see ERRORS below for meanings). This call should never return.

compile()

The syntax of the compile() routine is as follows:

```
compile(instring, expbuf, endbuf, eof)
```

The first parameter, instring, is never used explicitly by the compile() routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of (char *)0 for this parameter.

The next parameter, expbuf, is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter endbuf is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (endbuf-expbuf) bytes, a call to ERROR(50) is made.

The parameter eof is the character which marks the end of the regular expression. This character is usually a /.

Each program that includes the <regexp.h> header file must have a #define statement for INIT. It is used for dependent declarations and initializations. Most often it is used to set a register variable to point to the beginning of the regular expression so that this register variable can be used in the declarations for GETC, PEEKC, and UNGETC. Otherwise it can be used to declare external variables that might be used by GETC, PEEKC and UNGETC. (See examples below.)

step(), advance()

The first parameter to the step() and advance() functions is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter, expbuf, is the compiled regular expression which was obtained by a call to the function compile().

The function step() returns non-zero if some substring of string matches the regular expression in expbuf and 0 if

there is no match. If there is a match, two external character pointers are set as a side effect to the call to step(). The variable loc1 points to the first character that matched the regular expression; the variable loc2 points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire input string, loc1 will point to the first character of string and loc2 will point to the null at the end of string.

The function advance() returns non-zero if the initial substring of string matches the regular expression in expbuf. If there is a match, an external character pointer, loc2, is set as a side effect. The variable loc2 points to the next character in string after the last character that matched.

When advance() encounters a * or \{ \} sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, advance() will back up along the string until it finds a match or reaches the point in the string that initially matched the * or \{ \}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer locs is equal to the point in the string at sometime during the backing up process, advance() will break out of the loop that backs up and will return zero.

The external variables circf, sed, and nbra are reserved.

EXAMPLES

The following is an example of how the regular expression macros and calls might be defined by an application program:

```
#define INIT         register char *sp = instring;
#define GETC       (*sp++)
#define PEEKC      (*sp)
#define UNGETC(c)    (--sp)
#define RETURN(*c)    return;
#define ERROR(c)     regerr
#include <regexp.h>
 . . .
     (void) compile(*argv, expbuf, &expbuf[ESIZE],'\0');
 . . .
```

```
        if (step(linebuf, expbuf))
                        succeed;
```

DIAGNOSTICS

The function compile() uses the macro RETURN on success and the macro ERROR on failure (see above). The functions step() and advance() return non-zero on a successful match and zero if there is no match. Errors are:

11 range endpoint too large.

16 bad number.

25 \ digit out of range.

36 illegal or missing delimiter.

41 no remembered search string.

42 \(\) imbalance.

43 too many \(.

44 more than 2 numbers given in \{ \}.

45 } expected after \.

46 first number exceeds second in \{ \}.

49 [ ] imbalance.

50 regular expression overflow.

SEE ALSO

regex(5)

rm

rm, rmdir - remove directory entries

SYNOPSIS

```
/usr/bin/rm [-f] [-i] file...
/usr/bin/rm -rR [-f] [-i] dirname...[file...]

/usr/xpg4/bin/rm [ -fiRr ] file...

/usr/bin/rmdir [-ps] dirname...
```

DESCRIPTION

/usr/bin/rm /usr/xpg4/bin/rm

The rm utility removes the directory entry specified by each file argument. If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with y (for yes), the file is deleted, otherwise the file remains.

If file is a symbolic link, the link will be removed, but the file or directory to which it refers will not be deleted. Users do not need write permission to remove a symbolic link, provided they have write permissions in the directory.

If multiple files are specified and removal of a file fails for any reason, rm will write a diagnostic message to standard error, do nothing more to the current file, and go on to any remaining files.

If the standard input is not a terminal, the utility will operate as if the -f option is in effect.

/usr/bin/rmdir

The rmdir utility will remove the directory entry specified by each dirname operand, which must refer to an empty directory.

Directories will be processed in the order specified. If a directory and a subdirectory of that directory are specified in a single invocation of rmdir, the subdirectory must be

specified before the parent directory so that the parent directory will be empty when rmdir tries to remove it.

OPTIONS

The following options are supported for /usr/bin/rm and /usr/xpg4/bin/rm:

-r Recursively remove directories and subdirectories in the argument list. The directory will be emptied of files and removed. The user is normally prompted for removal of any write-protected files which the directory contains. The write-protected files are removed without prompting, however, if the -f option is used, or if the standard input is not a terminal and the -i option is not used.

Symbolic links that are encountered with this option will not be traversed.

If the removal of a non-empty, write-protected directory is attempted, the utility will always fail (even if the -f option is used), resulting in an error message.

-R Same as -r option.

/usr/bin/rm

The following options are supported for /usr/bin/rm only:

-f Remove all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory is attempted, this option will not suppress an error message.

-i Interactive. With this option, rm prompts for confirmation before removing any files. It overrides the -f option and remains in effect even if the standard input is not a terminal.

/usr/xpg4/bin/rm

The following options are supported for /usr/xpg4/bin/rm only:

-f Do not prompt for confirmation. Do not write diagnostic messages or modify the exit status in the case of non-existent operands. Any previous occurences of the -i option will be ignored.

-i Prompt for confirmation. Any occurences of the -f option will be ignored.

/usr/bin/rmdir

The following options are supported for /usr/bin/rmdir only:

-p Allow users to remove the directory dirname and its parent directories which become empty. A message is printed on the standard error about whether the whole path is removed or part of the path remains for some reason.

-s Suppress the message printed on the standard error when -p is in effect.

OPERANDS

The following operands are supported:

file A path name of a directory entry to be removed.

dirname A path name of an empty directory to be removed.

USAGE

See largefile(5) for the description of the behavior of rm and rmdir when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

/usr/bin/rm /usr/xpg4/bin/rm

The following command:

```
example% rm a.out core
```

removes the directory entries: a.out and core.

The following command:

```
example% rm -rf junk
```

removes the directory junk and all its contents, without prompting.

/usr/bin/rmdir
If a directory a in the current directory is empty except it contains a directory b and a/b is empty except it contains a directory c,

```
example% rmdir -p a/b/c
```

will remove all three directories.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of rm and rmdir: LC_COLLATE, LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 If the -f option was not specified, all the named directory entries were removed; otherwise, all the existing named directory entries were removed.

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/rm /usr/bin/rmdir

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	enabled

/usr/xpg4/bin/rm

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	enabled

SEE ALSO

rmdir(2), unlink(2), attributes(5), environ(5), large-

file(5), xpg4(5)

DIAGNOSTICS

All messages are generally self-explanatory.

It is forbidden to remove the files "." and ".." in order to avoid the consequences of inadvertently doing something like the following:

```
rm -r .*
```

NOTES

A -- permits the user to mark explicitly the end of any command line options, allowing rm to recognize file arguments that begin with a -. As an aid to BSD migration, rm will accept - as a synonym for --. This migration aid may disappear in a future release. If a -- and a - both appear on the same command line, the second will be interpreted as a file.

script

script - make record of a terminal session

SYNOPSIS

script [-a] [filename]

DESCRIPTION

script makes a record of everything printed on your screen. The record is written to filename. If no file name is given, the record is saved in the file typescript.

The script command forks and creates a sub-shell, according to the value of $SHELL, and records the text from this session. The script ends when the forked shell exits or when CTRL-D is typed.

OPTIONS

-a Append the session record to filename, rather than overwrite it.

NOTES

script places everything that appears on the screen in filename, including prompts.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	enabled

SEE ALSO

attributes(5)

sed

sed - stream editor

SYNOPSIS

```
/usr/bin/sed [ -n ] script [ file ... ]
/usr/bin/sed [ -n ] [ -e script ] ...
     [ -f script_file ] ... [ file ... ]

/usr/xpg4/bin/sed [ -n ] script [ file ... ]
/usr/xpg4/bin/sed [ -n ] [ -e script ] ...
     [ -f script_file ] ... [ file ... ]
```

DESCRIPTION

The sed utility is a stream editor that reads one or more text files, makes editing changes according to a script of editing commands, and writes the results to standard output. The script is obtained from either the script operand string, or a combination of the option-arguments from the -e script and -f script_file options.

The sed utility is a text editor. It cannot edit binary files or files containing ASCII NUL (\0) characters or very long lines.

OPTIONS

The following options are supported;

-e script — script is an edit command for sed. See usage below for more information on the format of script. If there is just one -e option and no -f options, the flag -e may be omitted.

-f script_file — Take the script from script_file. script_file consists of editing commands, one per line.

-n — Suppress the default output.

Multiple -e and -f options may be specified. All commands are added to the script in the order specified, regardless of their origin.

OPERANDS

The following operands are supported:

file — A path name of a file whose contents will be read and edited. If multiple file operands are specified, the named files will be read in the order specified and the concatenation will be edited. If no file operands are specified, the standard input will be used.

script — A string to be used as the script of editing commands. The application must not present a script that violates the restrictions of a text file except that the final character need not be a NEWLINE character.

USAGE

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

```
[ address [ , address ] ] command [ arguments ]
```

Zero or more blank characters are accepted before the first address and before command. Any number of semicolons are accepted before the first address.

In normal operation, sed cyclically copies a line of input (less its terminating NEWLINE character) 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 copies the resulting pattern space to the standard output (except under -n) and deletes the pattern space. Whenever the pattern space is written to standard output or a named file, sed will immediately follow it with a NEWLINE character.

Some of the commands use a hold space to save all or part of the pattern space for subsequent retrieval. The pattern and hold spaces will each be able to hold at least 8192 bytes.

sed Addresses

An address is either empty, a decimal number that counts input lines cumulatively across files, a $ that addresses the last line of input, or a context address, which consists of a /regular expression/ as described on the regexp(5) manual page.

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 address. Thereafter the process is repeated, looking again for the first address. (If the second address is a number less than or equal to the line number selected by the first address, only the line corresponding to the first address is selected.)
Typically, address are separated from each other by a comma (,). They may also be separated by a semicolon (;).

sed Regular Expressions

sed supports the basic regular expressions described on the regexp(5) manual page, with the following additions:

\cREc — In a context address, the construction \cREc, where c is any character other than a backslash or NEWLINE character, is identical to /RE/. If the character designated by c appears following a backslash, then it is considered to be that literal character, which does not terminate the RE. For example, in the context address \xabc\xdefx, the second x stands for itself, so that the regular expression is abcxdef.

\n — The escape sequence \n matches a NEWLINE character embedded in the pattern space. A literal NEWLINE character must not be used in the regular expression of a context address or in the substitute command.

Editing commands can be applied only to non-selected pattern spaces by use of the negation command ! (described below).

sed Editing Commands

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

The r and w commands take an optional rfile (or wfile) parameter, separated from the command letter by one or more

blank characters.

Multiple commands can be specified by separating them with a semicolon (;) on the same command line.

The text argument consists of one or more lines, all but the last of which end with \ to hide the NEWLINE. Each embedded NEWLINE character in the text must be preceded by a backslash. Other backslashes in text are removed and the following character is treated literally. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The rfile or wfile argument must terminate the command line and must be preceded by exactly one blank. The use of the wfile parameter causes that file to be initially created, if it does not exist, or will replace the contents of an existing file. There can be at most 10 distinct wfile arguments.
Regular expressions match entire strings, not just individual lines, but a NEWLINE character is matched by \n in a sed RE; a NEWLINE character is not allowed in an RE. Also note that \n cannot be used to match a NEWLINE character at the end of an input line; NEWLINE characters appear in the pattern space as a result of the N editing command.

Two of the commands take a command-list, which is a list of sed commands separated by NEWLINE characters, as follows:

```
{ command
command
}
```

The { can be preceded with blank characters and can be followed with white space. The commands can be preceded by white space. The terminating } must be preceded by a NEWLINE character and can be preceded or followed by <blank>s. The braces may be preceded or followed by <blank>s. The command may be preceeded by <blank>s, but may not be followed by <blank>s.

The following table lists the functions.

Maximum Number of Addresses	Command	Description
2	{command-list	

	}	Execute command-list only when the pattern space is selected.
1	a\ text	Append by executing N command or beginning a new cycle. Place text on the output before reading the next input line.
2	b label	Branch to the : command bearing the label. If label is empty, branch to the end of the script. Labels are recognized unique up to eight characters.
2	c\ text	Change. Delete the pattern space. Place text on the output. Start the next cycle.
2	d	Delete the pattern space. Start the next cycle.
2	D	Delete the initial segment of the pattern space through the first new-line. Start the next cycle. (See the N command below.)
2	g	Replace the contents of the pattern space by the contents of the hold space.
2	G	Append the contents of the hold space to the pattern space.
2	h	Replace the contents of the hold space by the contents of the pattern space.
2	H	Append the contents of the pattern space to the hold space.
1	i\ text	Insert. Place text on the standard output.
2	l	usr/bin/sed: List the pattern space on the standard output in an unambiguous form. Non-printable characters are displayed in

		octal notation and long lines are folded. usr//xpg4/bin/sed: List the pattern space on the standard output in an unambiguous form. Non-printable characters are displayed in octal notation and long lines are folded. The characters (\\, \a, \b, \f, \r, \t, and \v) are written as the corresponding escape sequences. Non-printable characters not in that table will be written as one three-digit octal number (with a preceding backslash character) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than nine bits, the format used for non-printable characters is implementation-dependent. Long lines will be folded, with the point of folding indicated by writing a backslash followed by a newline character; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line will be marked with a $.
2	n	Copy the pattern space to the standard output if default is not suppressed. Replace the pattern space with the next line of input.
2	N	Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.) If the end of input is reached, the N command verb shall branch to the end of the script and quit without starting a new cycle.
2	p	Print. Copy the pattern space to the standard output.

Maximum Number of Addresses	Command	Description
2	P	Copy the initial segment of the pattern space through the first new-line to the standard output.
1	q	Quit. Branch to the end of the script. Do not start a new cycle.
2	r rfile	Read the contents of rfile. Place them on the output before reading the next input line.
2	t label	Test. Branch to the : command bearing the label if any substitutions have been made since the most recent reading of an input line or execution of a t. If label is empty, branch to the end of the script.
2	w wfile	Write. Append the pattern space to wfile. The first occurrence of w will cause wfile to be cleared. Subsequent invocations of w will append. Each time the sed command is used, wfile is overwritten.
2	x	Exchange the contents of the pattern and hold spaces.
2	! command	Don't. Apply the command (or group, if command is {) only to lines not selected by the address(es).
0	: label	This command does nothing; it bears a label for b and t commands to branch to.
1	=	Place the current line number on the standard output as a line.
2	{	Execute the following commands through

		a matching } only when the pattern space is selected.
0		An empty command is ignored.
0	#	If a # appears as the first character on a line of a script file, then that entire line is treated as a comment, with one exception: if a # appears on the first line and the character after the # is an n, then the default output will be suppressed. The rest of the line after #n is also ignored. A script file must contain at least one non-comment line.

Maximum Number of Addresses	Command (Using strings) and Description
2	s/regular expression/replacement/flags Substitute the replacement string for instances of the regular expression in the pattern space. Any character other than backslash or newline can be used instead of a slash to delimit the RE and the replacement. Within the RE and the replacement, the RE delimiter itself can be used as a literal character if it is preceded by a backslash. An ampersand (&) appearing in the replacement will be replaced by the string matching the RE. The special meaning of & in this context can be suppressed by preceding it by backslash. The characters \n, where n is a digit, will be replaced by the text matched by the corresponding backreference expression. For each backslash (\) encountered in scanning replacement from beginning to end, the following character loses its special meaning (if any). It is unspecified what special meaning is given to any character other than &, \ or digits. A line can be split by substituting a newline character into it. The application must escape the newline character in the replacement by

preceding it by backslash. A substitution is considered to have been performed even if the replacement string is identical to the string that it replaces. For a fuller description see ed(1). flags is zero or more of:

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

g
Global. Substitute for all nonoverlapping instances of the regular expression rather than just the first one. If both g and n are specified, the results are unspecified.

p
Print the pattern space if a replacement was made.

w wfile
Write. Append the pattern space to wfile if a replacement was made. The first occurrence of w will cause wfile to be cleared. Subsequent invocations of w will append. Each time the sed command is used, wfile is overwritten.

2 y/string1/string2/
Transform. Replace all occurrences of characters in string1 with the corresponding characters in string2. string1 and string2 must have the same number of characters, or if any of the characters in string1 appear more than once, the results are undefined. Any character other than backslash or newline can be used instead of slash to delimit the strings. Within string1 and string2, the delimiter itself can be used as a literal character if it is preceded by a backslash. For example, y/abc/ABC/ replaces a with A, b with B, and c with C.

See largefile(5) for the description of the behavior of sed when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

This sed script simulates the BSD cat -s command, squeezing excess blank lines from standard input.

```
sed -n '
# Write non-empty lines.
/./     {
        p
        d
        }
# Write a single empty line, then look for more empty
lines.
/^$/        p
# Get next line, discard the held <newline> (empty
line),
# and look for more empty lines.
:Empty
/^$/        {
        N
        s/.//
        b Empty
        }
# Write the non-empty line before going back to search
# for the first in a set of empty lines.
        p
'
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of sed: LC_COLLATE, LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/sed

ATTRIBUTE TYPE	ATTRIBUTE VALUE

Availability	SUNWcsu
CSI	Not enabled

/usr/xpg4/bin/sed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	Enabled

SEE ALSO

awk(1), ed(1), grep(1), attributes(5), environ(5), large-file(5), regexp(5), xpg4(5)

shutdown

shutdown - shut down system, change system state

SYNOPSIS

```
/usr/sbin/shutdown [ -y ] [ -g grace-period ]
     [ -i init-state ] [ message ]
```

DESCRIPTION

shutdown is executed by the super-user to change the state of the machine. In most cases, it is used to change from the multi-user state (state 2) to another state.

By default, shutdown brings the system to a state where only the console has access to the operating system. This state is called single-user.

Before starting to shut down daemons and killing processes, shutdown sends a warning message and, by default, a final message asking for confirmation. message is a string that is sent out following the standard warning message "The system will be shut down in ..." If the string contains more than one word, it should be contained within single (') or double (") quotation marks.

The warning message and the user provided message are output when there are 7200, 3600, 1800, 1200, 600, 300, 120, 60, and 30 seconds remaining before shutdown begins. See EXAMPLES.

System state definitions are:

state 0 — Stop the operating system.

state 1 — State 1 is referred to as the administrative state. In state 1 file systems required for multi-user operations are mounted, and logins requiring access to multi-user file systems can be used. When the system comes up from firmware mode into state 1, only the console is active and other multi-user (state 2) services are unavailable. Note that not all user processes are stopped when transitioning from multi-user state to state 1.

state s, S
State s (or S) is referred to as the single-user state. All user processes are stopped on transitions to this state. In the single-user state, file systems required for multi-user logins are unmounted and the system can only be accessed through the console. Logins requiring access to multi-user file systems cannot be used.

state 5
Shut the machine down so that it is safe to remove the power. Have the machine remove power, if possible. The rc0 procedure is called to perform this task.

state 6
Stop the operating system and reboot to the state defined by the initdefault entry in /etc/inittab. The rc6 procedure is called to perform this task.

OPTIONS

-y
Pre-answer the confirmation question so the command can be run without user intervention.

-g grace-period
Allow the super-user to change the number of seconds from the 60-second default.

-i init-state
If there are warnings, init-state specifies the state init is to be in. By default, system state \ >qspq is used.

EXAMPLES

In the following example, shutdown is being executed on host foo and is scheduled in 120 seconds. The warning message is output 2 minutes, 1 minute, and 30 seconds before the final confirmation message.

```
example# shutdown -i S -g 120 "===== disk replacement
====="
Shutdown started.   Tue Jun   7  14:51:40 PDT  1994
Broadcast Message from root (pts/1) on foo Tue Jun  7
14:51:41...
The system will be shut down in 2 minutes
===== disk replacement =====
```

```
          Broadcast Message from root (pts/1) on foo Tue Jun  7
14:52:41...
          The system will be shut down in 1 minutes
          ===== disk replacement =====
          Broadcast Message from root (pts/1) on foo Tue Jun  7
14:53:41...
          The system will be shut down in 30 seconds
          ===== disk replacement =====
          Do you want to continue? (y or n):

FILES
     /etc/inittab        controls process dispatching by init
```

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

SEE ALSO

boot(1M), halt(1M), init(1M), killall(1M), reboot(1M), ufsdump(1M), init.d(4), inittab(4), nologin(4), attributes(5)

sleep

sleep - suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

The sleep utility will suspend execution for at least the integral number of seconds specified by the time operand.

OPERANDS

The following operands are supported:

time A non-negative decimal integer specifying the number of seconds for which to suspend execution.

EXAMPLES

To execute a command after a certain amount of time:

```
(sleep 105; command)&
```

or to execute a command every so often:

```
while true
do
     command
     sleep 37
done
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of sleep: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 The execution was successfully suspended for at least time seconds, or a SIGALRM signal was received (see NOTES).

>0 An error has occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

SEE ALSO

wait(1), alarm(2), sleep(3C), wait(3B), attributes(5), environ(5)

NOTES

If the sleep utility receives a SIGALRM signal, one of the following actions will be taken:

+ Terminate normally with a zero exit status.

+ Effectively ignore the signal.

The sleep utility will take the standard action for all other signals.

sort

sort - sort, merge, or sequence check text files

SYNOPSIS

```
/usr/bin/sort [ -cmu ] [ -o output ] [ -T directory ]
     [ -y [ kmem ]] [ -z recsz ]  [  -dfiMnr  ]  [  -b  ]  [
-t char ]
     [ -k keydef ] [ +pos1 [ -pos2 ]] [ file...]

/usr/xpg4/bin/sort [ -cmu ] [ -o output ]
     [ -T directory ] [ -y [ kmem ]] [ -z recsz ] [  -dfiMnr
] [ -b ] [ -t char ]
     [ -k keydef ] [ +pos1 [ -pos2 ]] [ file...]
```

DESCRIPTION

The sort command sorts lines of all the named files together and writes the result on the standard output.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line. Lines are ordered according to the collating sequence of the current locale.

OPTIONS

The following options alter the default behavior:

/usr/bin/sort

-c Check that the single input file is ordered as specified by the arguments and the collating sequence of the current locale. The exit code is set and no output is produced unless the file is out of sort.

/usr/xpg4/bin/sort

-c Same as /usr/bin/sort except no output is produced under any circumstances.

-m Merge only. The input files are assumed to be already sorted.

-u Unique: suppress all but one in each set of lines having equal keys. If used with the -c option,

check that there are no lines with duplicate keys in addition to checking that the input file is sorted.

-o output Specify the name of an output file to be used instead of the standard output. This file can be the same as one of the input files.

-T directory
The directory argument is the name of a directory in which to place temporary files.

-y kmem The amount of main memory initially used by sort. If this option is omitted, sort begins using a system default memory size, and continues to use more space as needed. If kmem is present, sort will start using that number of Kbytes of memory, unless the administrative minimum or maximum is exceeded, in which case the corresponding extremum will be used. Thus, -y 0 is guaranteed to start with minimum memory. -y with no kmem argument starts with maximum memory.

-z recsz (obsolete). This option was used to prevent abnormal termination when lines longer than the system-dependent default buffer size are encountered. Because sort automatically allocates buffers large enough to hold the longest line, this option has no effect.

Ordering options
The following options override the default ordering rules. When ordering options appear independent of any key field specifications, the requested field ordering rules are applied globally to all sort keys. When attached to a specific key (see Sort Key Options), the specified ordering options override all global ordering options for that key. In the obsolescent forms, if one or more of these options follows a +pos1 option, it will affect only the key field specified by that preceding option.

-d ``Dictionary'' order: only letters, digits, and blanks (spaces and tabs) are significant in comparisons.

-f Fold lower-case letters into upper case.

-i Ignore non-printable characters.

-M Compare as months. The first three non-blank characters of the field are folded to upper case and compared. For example, in English the sorting order is "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The -M option implies the -b option (see below).

-n Restrict the sort key to an initial numeric string, consisting of optional blank characters, optional minus sign, and zero or more digits with an optional radix character and thousands separators (as defined in the current locale), which will be sorted by arithmetic value. An empty digit string is treated as zero. Leading zeros and signs on zeros do not affect ordering.

-r Reverse the sense of comparisons.

Field Separator options

The treatment of field separators can be altered using the following options:

-b Ignore leading blank characters when determining the starting and ending positions of a restricted sort key. If the -b option is specified before the first sort key option, it is applied to all sort key options. Otherwise, the -b option can be attached independently to each -k field_start, field_end, or +pos1 or -pos2 option-argument (see below).

-t char Use char as the field separator character. char is not considered to be part of a field (although it can be included in a sort key). Each occurrence of char is significant (for example, <char><char> delimits an empty field). If -t is not specified, blank characters are used as default field separators; each maximal non-empty sequence of blank characters that follows a non-blank character is a field separator.

Sort Key options

Sort keys can be specified using the options:

-k keydef The keydef argument is a restricted sort key field

definition. The format of this definition is:

```
-k field_start [ type ] [ ,field_end [ type ] ]
]
```

where:

field_start and field_end
: define a key field restricted to a portion of the line.

type
: is a modifier from the list of characters bdfiMnr. The b modifier behaves like the -b option, but applies only to the field_start or field_end to which it is attached and characters within a field are counted from the first non-blank character in the field. (This applies separately to first_character and last_character.) The other modifiers behave like the corresponding options, but apply only to the key field to which they are attached. They have this effect if specified with field_start, field_end or both. If any modifier is attached to a field_start or to a field_end, no option applies to either.

When there are multiple key fields, later keys are compared only after all earlier keys compare equal. Except when the -u option is specified, lines that otherwise compare equal are ordered as if none of the options -d, -f, -i, -n or -k were present (but with -r still in effect, if it was specified) and with all bytes in the lines significant to the comparison.

The notation:

```
-k field_start[type][,field_end[type]]
```

defines a key field that begins at field_start and ends at field_end inclusive, unless field_start falls beyond the end of the line or after field_end, in which case the key field is empty. A missing field_end means the last character of the line.

A field comprises a maximal sequence of non-separating characters and, in the absence of option -t, any preceding field separator.

The field_start portion of the keydef option-argument has the form:

field_number[.first_character]

Fields and characters within fields are numbered starting with 1. field_number and first_character, interpreted as positive decimal integers, specify the first character to be used as part of a sort key. If .first_character is omitted, it refers to the first character of the field.

The field_end portion of the keydef option-argument has the form:

field_number[.last_character]

The field_number is as described above for field_start. last_character, interpreted as a non-negative decimal integer, specifies the last character to be used as part of the sort key. If last_character evaluates to zero or .last_character is omitted, it refers to the last character of the field specified by field_number.

If the -b option or b type modifier is in effect, characters within a field are counted from the first non-blank character in the field. (This applies separately to first_character and last_character.)

[+pos1[-pos2]]

(obsolete). Provide functionality equivalent to the -k keydef option.

pos1 and pos2 each have the form m.n optionally followed by one or more of the flags bdfiMnr. A starting position specified by +m.n is interpreted to mean the n+1st character in the m+1st field. A missing .n means .0, indicating the first character of the m+1st field. If the b flag is in effect n is counted from the first non-blank in

the m+1st field; +m.0b refers to the first non-blank character in the m+1st field.

A last position specified by -m.n is interpreted to mean the nth character (including separators) after the last character of the mth field. A missing .n means .0, indicating the last character of the mth field. If the b flag is in effect n is counted from the last leading blank in the m+1st field; -m.1b refers to the first non-blank in the m+1st field.

The fully specified +pos1 -pos2 form with type modifiers T and U:

```
+w.xT -y.zU
```

is equivalent to:

```
undefined            (z==0 & U contains b & -t is present)
-k w+1.x+1T,y.0U     (z==0 otherwise)
-k w+1.x+1T,y+1.zU   (z > 0)
```

Implementations support at least nine occurrences of the sort keys (the -k option and obsolescent +pos1 and -pos2) which are significant in command line order. If no sort key is specified, a default sort key of the entire line is used.

OPERANDS

The following operand is supported:

file A path name of a file to be sorted, merged or checked. If no file operands are specified, or if a file operand is -, the standard input will be used.

USAGE

See largefile(5) for the description of the behavior of sort when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

In the following examples, non-obsolescent and obsolescent ways of specifying sort keys are given as an aid to understanding the relationship between the two forms.

Either of the following commands sorts the contents of infile with the second field as the sort key:

```
example% sort -k 2,2 infile
example% sort +1 -2 infile
```

Either of the following commands sorts, in reverse order, the contents of infile1 and infile2, placing the output in outfile and using the second character of the second field as the sort key (assuming that the first character of the second field is the field separator):

```
example% sort -r -o outfile -k 2.2,2.2 infile1 infile2
example% sort -r -o outfile +1.1 -1.2 infile1 infile2
```

Either of the following commands sorts the contents of infile1 and infile2 using the second non-blank character of the second field as the sort key:

```
example% sort -k 2.2b,2.2b infile1 infile2
example% sort +1.1b -1.2b infile1 infile2
```

Either of the following commands prints the passwd(4) file (user database) sorted by the numeric user ID (the third colon-separated field):

```
example% sort -t : -k 3,3n /etc/passwd
example% sort -t : +2 -3n /etc/passwd
```

Either of the following commands prints the lines of the already sorted file infile, suppressing all but one occurrence of lines having the same third field:

```
example% sort -um -k 3.1,3.0 infile
example% sort -um +2.0 -3.0 infile
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of sort: LC_COLLATE, LC_MESSAGES, and NLSPATH.

LC_CTYPE — Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- versus multi-byte characters in arguments and input files) and the behavior of character classification for

the -b, -d, -f, -i and -n options.

LC_NUMERIC Determine the locale for the definition of the radix character and thousands separator for the -n option.

EXIT STATUS

The following exit values are returned:

0 All input files were output successfully, or -c was specified and the input file was correctly sorted.

1 Under the -c option, the file was not ordered as specified, or if the -c and -u options were both specified, two input lines were found with equal keys.

>1 An error occurred.

FILES

/var/tmp/stm??? temporary files

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/sort

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWesu
CSI	Enabled

/usr/xpg4/bin/sort

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	Enabled

SEE ALSO comm(1), join(1), uniq(1), passwd(4), attributes(5),

|environ(5), lar| gefile(5), xpg4(5)|
| | |
DIAG| NOSTICS | |
|Comments and ex| its with non-zero | status for various trouble
|conditions (for|example, when inp| ut lines are too long), and
|for disorders d| iscovered under th| e -c option.
| | |
NOTE| S | |
|When the last l| ine of an input fi| le is missing a new-line
|character, sor| t appends one, pr| ints a warning message, and
|continues. | |
| | |
|sort does not g| uarantee preservat| ion of relative line order-
|ing on equal ke| ys. |

spell

spell, hashmake, spellin, hashcheck - report spelling errors

SYNOPSIS

```
spell [ -bilvx ]
     [ +local_file ] [ file]...

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list
```

DESCRIPTION

The spell command collects words from the named files and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, or suffixes) from words in the spelling list are written to the standard output.

If there are no file arguments, words to check are collected from the standard input. spell ignores most troff(1), tbl(1), and eqn(1) constructs. Copies of all output words are accumulated in the history file (spellhist), and a stop list filters out misspellings (for example, their=thy-y+ier) that would otherwise pass.

By default, spell (like deroff(1)) follows chains of included files (.so and .nx troff(1) requests), unless the names of such included files begin with /usr/lib.

The standard 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.

Three programs help maintain and check the hash lists used by spell:

hashmake — Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.

spellin Reads n hash codes from the standard input and writes a compressed spelling list on the standard output.

hashcheck Reads a compressed spelling_list and recreates the nine-digit hash codes for all the words in it. It writes these codes on the standard output.

OPTIONS

The following options are supported:

-b Check British spelling. Besides preferring "centre," "colour," "programme," "speciality," "travelled," and so forth, this option insists upon -ise in words like "standardise."

-i Cause deroff(1) to ignore .so and .nx commands. If deroff(1) is not present on the system, then this option is ignored.

-l Follow the chains of all included files.

-v Print all words not literally in the spelling list, as well as plausible derivations from the words in the spelling list.

-x Print every plausible stem, one per line, with = preceding each word.

+local_file Specify a set of words that are correct spellings (in addition to spell's own spelling list) for each job. local_file is the name of a user-provided file that contains a sorted list of words, one per line. Words found in local_file are removed from spell's output. Use sort(1) to order local_file in ASCII collating sequence. If this ordering is not followed, some entries in local_file may be ignored.

OPERANDS

The following operands are supported:

file A path name of a text file to check for spelling errors. If no files are named, words are collected from the standard input.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of spell: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/var/adm/spellhist	history file
/usr/share/lib/dict/words	master dictionary

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWesu

SEE ALSO

deroff(1), eqn(1), sort(1), tbl(1), troff(1), attributes(5), environ(5)

NOTES

Misspelled words can be monitored by default by setting the H_SPELL variable in /usr/bin/spell to the name of a file that has permission mode 666.

spell works only on English words defined in the U.S. ASCII codeset.

Because copies of all output are accumulated in the spellhist file, spellhist may grow quite large and require purging.

BUGS

The spelling list's coverage is uneven; new installations may wish to monitor the output for several months to gather local additions.

British spelling was done by an American.

sum

sum - print checksum and block count for a file

SYNOPSIS

sum [-r] [file...]

DESCRIPTION

The sum utility calculates and prints a 16-bit checksum for the named file and the number of 512-byte blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

OPTIONS

The following options are supported:

-r Use an alternate (machine-dependent) algorithm in computing the checksum.

OPERANDS

The following operands are supported:

file A path name of a file. If no files are named, the standard input is used.

USAGE

See largefile(5) for the description of the behavior of sum when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of sum: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned.

0 Successful completion.

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWesu
CSI	enabled

SEE ALSO

cksum(1), sum(1B), wc(1), attributes(5), environ(5), largefile(5)

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NOTES

Portable applications should use cksum(1).

sum and usr/ucb/sum (see sum(1B)) return different checksums.

tar

tar - create tape archives and add or extract files

SYNOPSIS

```
tar c [bBefFhiklnopPqvwX [ 0-7 ]] [ block ] [ tarfile ]
     [ exclude-file ] { -I include-file |
     -C directory file | file } ...

tar r [ bBefFhiklnqvw [ 0-7 ]] [ block ]
     { -I include-file | -C directory file | file } ...

tar t [ BefFhiklnqvX [ 0-7 ]] [ tarfile ]
     [ exclude-file ] { -I include-file | file } ...

tar u [ bBefFhiklnqvw [ 0-7 ]] [ block ] [ tarfile ]
     file ...

tar x [ BefFhiklmnopqvwX [ 0-7 ]] [ tarfile ]
     [ exclude-file ] [ file ... ]
```

DESCRIPTION

The tar command archives and extracts files to and from a single file called a tarfile. A tarfile is usually a magnetic tape, but it can be any file. tar's actions are controlled by the key argument. The key is a string of characters containing exactly one function letter (c, r, t , u, or x) and zero or more function modifiers (letters or digits), depending on the function letter used. The key string contains no SPACE characters. Function modifier arguments are listed on the command line in the same order as their corresponding function modifiers appear in the key string.

The -I include-file, -C directory file, and file arguments specify which files or directories are to be archived or extracted. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory. Arguments appearing within braces ({ indicate that one of the arguments must be specified.

OPTIONS

The following options are supported:

-I include-file
Open include-file containing a list of files, one per line, and treat as if each file appeared separately on the command line. Be careful of trailing white spaces. In the case where excluded files (see X function modifier) are also specified, they take precedence over all included files. If a file is specified in both the exclude-file and the include-file (or on the command line), it will be excluded.

-C directory file
Perform a chdir (see cd(1)) operation on directory and perform the c (create) or r (replace) operation on file. Use short relative path names for file. If file is `.', archive all files in directory. This option enables archiving files from multiple directories not related by a close common parent.

OPERANDS

The following operands are supported:

file A path name of a regular file or directory to be archived (when the c, r or u functions are specified), extracted (x) or listed (t). When file is the path name of a directory, the action applies to all of the files and (recursively) subdirectories of that directory. The directory portion of file (see dirname(1)) cannot exceed 155 characters. The file name portion (see basename(1)) cannot exceed 100 characters.

Function Letters

The function portion of the key is specified by one of the following letters:

c Create. Writing begins at the beginning of the tarfile, instead of at the end.

r Replace. The named files are written at the end of the tarfile.

t Table of Contents. The names of the specified files are listed each time they occur in the tarfile. If no file argument is given, the names of all files in the tarfile are listed. With the v function modifier, additional information for the specified files is displayed.

u Update. The named files are written at the end of the

tarfile if they are not already in the tarfile, or if they have been modified since last written to that tarfile. An update can be rather slow. A tarfile created on a 5.x system cannot be updated on a 4.x system.

x Extract or restore. The named files are extracted from the tarfile and written to the directory specified in the tarfile, relative to the current directory. Use the relative path names of files and directories to be extracted. If a named file matches a directory whose contents has been written to the tarfile, this directory is recursively extracted. The owner, modification time, and mode are restored (if possible); otherwise, to restore owner, you must be the super-user. Character-special and block-special devices (created by mknod(1M)) can only be extracted by the super-user. If no file argument is given, the entire content of the tarfile is extracted. If the tarfile contains several files with the same name, each file is written to the appropriate directory, overwriting the previous one. Filename substitution wildcards cannot be used for extracting files from the archive; rather, use a command of the form:

```
tar xvf... /dev/rmt/0 `tar tf... /dev/rmt/0 |
grep 'pattern'`
```

When extracting tapes created with the r or u functions, directory modification times may not be set correctly. These same functions cannot be used with many tape drives due to tape drive limitations such as the absence of backspace or append capabilities.

When using the r, u, or x functions or the X function modifier, the named files must match exactly the corresponding files in the tarfile. For example, to extract ./thisfile, you must specify ./thisfile, and not thisfile. The t function displays how each file was archived.

Function Modifiers

The characters below may be used in conjunction with the letter that selects the desired function.

b Blocking Factor. Use when reading or writing to raw magnetic archives (see f below). The block argument specifies the number of 512-byte tape blocks to be included in each read or write operation performed on

the tarfile. The minimum is 1, the default is 20. The maximum value is a function of the amount of memory available and the blocking requirements of the specific tape device involved (see mtio(7I) for details.)

When a tape archive is being read, its actual blocking factor will be automatically detected, provided that it is less than or equal to the nominal blocking factor (the value of the block argument, or the default value if the b modifier is not specified). If the actual blocking factor is greater than the nominal blocking factor, a read error will result. See Example 5 in

B Block. Force tar to perform multiple reads (if necessary) to read exactly enough bytes to fill a block. This function modifier enables tar to work across the Ethernet, since pipes and sockets return partial blocks even when more data is coming. When reading from standard input, '-', this function modifier is selected by default to ensure that tar can recover from short reads.

e Error. Exit immediately with a positive exit status if any unexpected errors occur. The SYSV3 ENVIRONMENT variable overrides the default behavior. (See ENVIRONMENT section below.)

f File. Use the tarfile argument as the name of the tarfile. If f is specified, /etc/default/tar is not searched. If f is omitted, tar will use the device indicated by the TAPE environment variable, if set; otherwise, it will use the default values defined in /etc/default/tar. If the name of the tarfile is '-', tar writes to the standard output or reads from the standard input, whichever is appropriate. tar can be used as the head or tail of a pipeline. tar can also be used to move hierarchies with the command:

```
example% cd fromdir; tar cf - .
          | (cd todir; tar xfBp -)
```

F With one F argument, tar excludes all directories named SCCS and RCS from the tarfile. With two arguments, FF, tar excludes all directories named SCCS and RCS, all files with .o as their suffix, and all files named errs, core, and a.out. The SYSV3 environment variable overrides the default behavior. (See ENVIRONMENT sec-

tion below.)

h Follow symbolic links as if they were normal files or directories. Normally, tar does not follow symbolic links.

i Ignore directory checksum errors.

k size
Requires tar to use the size argument as the size of an archive in kilobytes. This is useful when the archive is intended for a fixed size device such as floppy disks. Large files are then split across volumes if they do not fit in the specified size.

l Link. Output error message if unable to resolve all links to the files being archived. If l is not specified, no error messages are printed.

m Modify. The modification time of the file is the time of extraction. This function modifier is valid only with the x function.

n The file being read is a non-tape device. Reading of the archive is faster since tar can randomly seek around the archive.

o Ownership. Assign to extracted files the user and group identifiers of the user running the program, rather than those on tarfile. This is the default behavior for users other than root. If the o function modifier is not set and the user is root, the extracted files will take on the group and user identifiers of the files on tarfile (see chown(1) for more information). The o function modifier is only valid with the x function.

p Restore the named files to their original modes, and ACLs if applicable, ignoring the present umask(1). This is the default behavior if invoked as super-user with the x function letter specified. If super-user, SETUID and sticky information are also extracted, and files are restored with their original owners and permissions, rather than owned by root. When this function modifier is used with the c function, ACLs are created in the tarfile along with other information. Errors will occur when a tarfile with ACLs is extracted

by previous versions of tar.

P Suppress the addition of a trailing "/" on directory entries in the archive.

q Stop after extracting the first occurrence of the named file. tar will normally continue reading the archive after finding an occurrence of a file.

v Verbose. Output the name of each file preceded by the function letter. With the t function, v provides additional information about the tarfile entries. The listing is similar to the format produced by the -l option of the ls(1) command.

w What. Output the action to be taken and the name of the file, then await the user's confirmation. If the response is affirmative, the action is performed; otherwise, the action is not performed. This function modifier cannot be used with the t function.

X Exclude. Use the exclude-file argument as a file containing a list of relative path names for files (or directories) to be excluded from the tarfile when using the functions c, x, or t. Be careful of trailing white spaces. Multiple X arguments may be used, with one exclude-file per argument. In the case where included files (see -I include-file option) are also specified, the excluded files take precedence over all included files. If a file is specified in both the exclude-file and the include-file (or on the command line), it will be excluded.

[0-7]
Select an alternative drive on which the tape is mounted. The default entries are specified in /etc/default/tar. If no digit or f function modifier is specified, the entry in /etc/default/tar with digit "0" is the default.

USAGE

See largefile(5) for the description of the behavior of tar when encountering files greater than or equal to 2 Gbyte (2**31 bytes). in the EXAMPLES below.

The automatic determination of the actual blocking factor

may be fooled when reading from a pipe or a socket (see the B function modifier below).

1/4" streaming tape has an inherent blocking factor of one 512-byte block. It can be read or written using any blocking factor.

This function modifier works for archives on disk files and block special devices, among others, but is intended principally for tape devices.

EXAMPLES

1. The following is an example using tar to create an archive of your home directory on a tape mounted on drive /dev/rmt/0:

```
example% cd
example% tar cvf /dev/rmt/0 .
messages from tar
```

   The c function letter means create the archive; the v function modifier outputs messages explaining what tar is doing; the f function modifier indicates that the tarfile is being specified (/dev/rmt/0 in this example). The dot (.) at the end of the command line indicates the current directory and is the argument of the f function modifier.

   Display the table of contents of the tarfile with the following command:

```
example% tar tvf /dev/rmt/0
```

   The output will be similar to the following for the POSIX locale:

```
rw-r--r-- 1677/40 2123  Nov  7 18:15 1985
./test.c
...
example%
```

   The columns have the following meanings:

   + column 1 is the access permissions to ./test.c
   + column 2 is the user-id/group-id of ./test.c
   + column 3 is the size of ./test.c in bytes
   + column 4 is the modification date of ./test.c. When the LC_TIME category is not set to the POSIX

```
            locale, a different format and date  order  field
            may be used.
         +  column 5 is the name of ./test.c
```

To extract files from the archive:

```
example% tar xvf /dev/rmt/0
messages from tar
example%
```

If there are multiple archive files on a tape, each is separated from the following one by an EOF marker. To have tar read the first and second archives from a tape with multiple archives on it, the non-rewinding version of the tape device name must be used with the f function modifier, as follows:

```
example% tar xvfp /dev/rmt/0n   read first archive
from tape
messages from tar
example% tar xvfp /dev/rmt/0n   read second archive
from tape
messages from tar
example%
```

Note that in some earlier releases, the above scenario did not work correctly, and intervention with mt(1) between tar invocations was necessary. To emulate the old behavior, use the non-rewind device name containing the letter b for BSD behavior. See the Close Operations section of the mtio(7I) manual page.

2. To archive files from /usr/include and from /etc to default tape drive 0:

```
example% tar c -C /usr  include -C /etc .
```

The table of contents from the resulting tarfile would produce output like the following:

```
include/
include/a.out.h
and all the other files in /usr/include ...
./chown
and all the other files in /etc
```

To extract all files in the include directory:

```
example% tar xv include
```

```
x include/, 0 bytes, 0 tape blocks
and all files under include...
```

3. The following is an example using tar to transfer files across the Ethernet. First, here is how to archive files from the local machine (example) to a tape on a remote system (host):

```
example% tar cvfb  -  20 files |
      rsh host dd of=/dev/rmt/0  obs=20b
messages from tar
example%
```

   In the example above, we are creating a tarfile with the c key letter, asking for verbose output from tar with the v function modifier, specifying the name of the output tarfile using the f function modifier (the standard output is where the tarfile appears, as indicated by the `-' sign), and specifying the blocksize (20) with the b function modifier. If you want to change the blocksize, you must change the blocksize arguments both on the tar command and on the dd command.

4. The following is an example that uses tar to retrieve files from a tape on the remote system back to the local system:

```
example% rsh -n host dd if=/dev/rmt/0 bs=20b |
     tar xvBfb - 20 files
messages from tar
example%
```

   In the example above, we are extracting from the tarfile with the x key letter, asking for verbose output from tar with the v function modifier, telling tar it is reading from a pipe with the B function modifier, specifying the name of the input tarfile using the f function modifier (the standard input is where the tarfile appears, as indicated by the `-' sign), and specifying the blocksize (20) with the b function modifier.

5. The following example creates an archive of the home directory on /dev/rmt/0 with an actual blocking factor of 19.

```
example% tar cvfb /dev/rmt/0 19 $HOME
```

   To recognize this archive's actual blocking factor

without using the b function modifier:

```
example% tar tvf /dev/rmt/0
tar: blocksize = 19
...
```

To recognize this archive's actual blocking factor using a larger nominal blocking factor:

```
example% tar tvf /dev/rmt/0 30
tar: blocksize = 19
...
```

Attempt to recognize this archive's actual blocking factor using a nominal blocking factor that is too small:

```
example% tar tvf /dev/rmt/0 10
tar: tape read error
```

ENVIRONMENT

SYSV3

This variable is used to override the default behavior of tar, provide compatibility with INTERACTIVE UNIX Systems and SCO UNIX installation scripts, and should not be used in new scripts. (It is intended for compatibility purposes only.) When set, the following options behave differently:

-F filename
: Uses filename to obtain a list of command line switches and files on which to operate.

-e
: Prevents files from being split across volumes. If there is insufficient room on one volume, tar prompts for a new volume. If the file will not fix on the new volume, tar exits with an error.

See environ(5) for descriptions of the following environment variables that affect the execution of tar: LC_CTYPE, LC_MESSAGES, LC_TIME, TZ, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0
: Successful completion.

>0
: An error occurred.

FILES

/dev/rmt/[0-7][b][n]
/dev/rmt/[0-7]l[b][n]
/dev/rmt/[0-7]m[b][n]
/dev/rmt/[0-7]h[b][n]
/dev/rmt/[0-7]u[b][n]
/dev/rmt/[0-7]c[b][n]
/etc/default/tar Settings may look like this:
archive0=/dev/rmt/0
archive1=/dev/rmt/0n
archive2=/dev/rmt/1
archive3=/dev/rmt/1n
archive4=/dev/rmt/0
archive5=/dev/rmt/0n
archive6=/dev/rmt/1
archive7=/dev/rmt/1n
/tmp/tar*

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled

SEE ALSO

ar(1), basename(1), cd(1), chown(1), cpio(1), csh(1), dirname(1), ls(1), mt(1), pax(1), setfacl(1), umask(1), mknod(1M), vold(1M), attributes(5), environ(5), largefile(5), mtio(7I)

DIAGNOSTICS

Diagnostic messages are output for bad key characters and tape read/write errors, and for insufficient memory to hold the link tables.

NOTES

There is no way to access for the n-th occurrence of a file.

Tape errors are handled ungracefully.

When the Volume Management daemon is running, accesses to floppy devices through the conventional device names (for example, /dev/rdiskette) may not succeed. See vold(1M) for further details.

The tar archive format allows UIDs and GIDs up to 2097151 to be stored in the archive header. Files with UIDs and GIDs greater than this value will be archived with the UID and GID of 60001.

tr

tr - translate characters

SYNOPSIS

```
/usr/bin/tr [-cs] string1 string2
/usr/bin/tr -s|-d [-c] string1
/usr/bin/tr -ds [-c] string1 string2

/usr/bin/xpg4/tr [-cs] string1 string2
/usr/bin/xpg4/tr -s|-d [-c] string1
/usr/bin/xpg4/tr -ds [-c] string1 string2
```

DESCRIPTION

The tr utility copies the standard input to the standard output with substitution or deletion of selected characters. The options specified and the string1 and string2 operands control translations that occur while copying characters and single-character collating elements.

OPTIONS

The following options are supported:

-c Complement the set of characters specified by string1.

-d Delete all occurrences of input characters that are specified by string1.

-s Replace instances of repeated characters with a single character.

When the -d option is not specified:

+ Each input character found in the array specified by string1 is replaced by the character in the same relative position in the array specified by string2. When the array specified by string2 is shorter that the one specified by string1, the results are unspecified.

+ If the -c option is specified, the complements of the characters specified by string1 (the set of all characters in the current character set, as defined by the

current setting of LC_CTYPE, except for those actually specified in the string1 operand) are placed in the array in ascending collation sequence, as defined by the current setting of LC_COLLATE.

+ Because the order in which characters specified by character class expressions or equivalence class expressions is undefined, such expressions should only be used if the intent is to map several characters into one. An exception is case conversion, as described previously.

When the -d option is specified:

+ Input characters found in the array specified by string1 will be deleted.

+ When the -c option is specified with -d, all characters except those specified by string1 will be deleted. The contents of string2 will be ignored, unless the -s option is also specified.

+ The same string cannot be used for both the -d and the -s option; when both options are specified, both string1 (used for deletion) and string2 (used for squeezing) are required.

When the -s option is specified, after any deletions or translations have taken place, repeated sequences of the same character will be replaced by one occurrence of the same character, if the character is found in the array specified by the last operand. If the last operand contains a character class, such as the following example:

```
tr -s '[:space:]'
```

the last operand's array will contain all of the characters in that character class. However, in a case conversion, as described previously, such as

```
tr -s '[:upper:]' '[:lower:]'
```

the last operand's array will contain only those characters defined as the second characters in each of the toupper or tolower character pairs, as appropriate. (See touopper(3C) and tolower(3C)).

An empty string used for string1 or string2 produces undefined results.

OPERANDS

The following operands are supported:

string1

string2 Translation control strings. Each string represents a set of characters to be converted into an array of characters used for the translation.

The operands string1 and string2 (if specified) define two arrays of characters. The constructs in the following list can be used to specify characters or single-character collating elements. If any of the constructs result in multi-character collating elements, tr will exclude, without a diagnostic, those multi-character elements from the resulting array.

character Any character not described by one of the conventions below represents itself.

\octal Octal sequences can be used to represent characters with specific coded values. An octal sequence consists of a backslash followed by the longest sequence of one-, two- or three-octal-digit characters (01234567). The sequence causes the character whose encoding is represented by the one-, two- or three-digit octal integer to be placed into the array. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading \ for each byte.

\character The backslash-escape sequences \a, \b, \f, \n, \r, \t, and \v are supported. The results of using any other character, other than an octal digit, following the backslash are unspecified.

/usr/xpg4/bin/tr
c-c
/usr/bin/tr
[c-c] Represents the range of collating elements between the range endpoints, inclusive, as

defined by the current setting of the LC_COLLATE locale category. The starting endpoint must precede the second endpoint in the current collation order. The characters or collating elements in the range are placed in the array in ascending collation sequence.

[:class:] Represents all characters belonging to the defined character class, as defined by the current setting of the LC_CTYPE locale category. The following character class names will be accepted when specified in string1:

alnum	blank	digit	lower	punct	upper
alpha	cntrl	graph	print	space	xdigit

In addition, character class expressions of the form [:name:] are recognized in those locales where the name keyword has been given a char-class definition in the LC_CTYPE category.
When both the -d and -s options are specified, any of the character class names will be accepted in string2. Otherwise, only character class names lower or upper are valid in string2 and then only if the corresponding character class upper and lower, respectively, is specified in the same relative position in string1. Such a specification is interpreted as a request for case conversion. When [:lower:] appears in string1 and [:upper:] appears in string2, the arrays will contain the characters from the toupper mapping in the LC_CTYPE category of the current locale. When [:upper:] appears in string1 and [:lower:] appears in string2, the arrays will contain the characters from the tolower mapping in the LC_CTYPE category of the current locale. The first character from each mapping pair will be in the array for string1 and the second character from each mapping pair will be in the array for string2 in the same relative position.

Except for case conversion, the characters specified by a character class expression are placed in the array in an unspecified order.

If the name specified for class does not define a valid character class in the current locale,

the behavior is undefined.

[=equiv=] Represents all characters or collating elements belonging to the same equivalence class as equiv, as defined by the current setting of the LC_COLLATE locale category. An equivalence class expression is allowed only in string1, or in string2 when it is being used by the combined -d and -s options. The characters belonging to the equivalence class are placed in the array in an unspecified order.

[x*n] Represents n repeated occurrences of the character x. Because this expression is used to map multiple characters to one, it is only valid when it occurs in string2. If n is omitted or is 0, it is interpreted as large enough to extend the string2-based sequence to the length of the string1-based sequence. If n has a leading 0, it is interpreted as an octal value. Otherwise, it is interpreted as a decimal value.

USAGE

See largefile(5) for the description of the behavior of tr when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

EXAMPLES

1. The following example creates a list of all words in file1 one per line in file2, where a word is taken to be a maximal string of letters.

```
tr -cs "[:alpha:]" "[\n*]" <file1 >file2
```

2. The next example translates all lower-case characters in file1 to upper-case and writes the results to standard output.

```
tr "[:lower:]" "[:upper:]" <file1
```

Note that the caveat expressed in the corresponding example in XPG3 is no longer in effect. This case conversion is now a special case that employs the tolower and toupper classifications, ensuring that proper mapping is accomplished (when the locale is correctly defined).

3. This example uses an equivalence class to identify accented variants of the base character e in file1, which are stripped of diacritical marks and written to file2.

```
tr "[=e=]" e <file1 >file2
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of tr: LC_COLLATE, LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 All input was processed successfully.

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/tr

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled

/usr/xpg4/bin/tr

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	Enabled

SEE ALSO

ed(1), sed(1), sh(1), tolower(3C), toupper(3C), ascii(5), attributes(5), environ(5), largefile(5), xpg4(5)

NOTES

Unlike some previous versions, the tr correctly processes NUL characters in its input stream. NUL characters can be stripped by using tr -d '\000'.

troff

troff - typeset or format documents

SYNOPSIS

troff [-a] [-f] [-Fdir] [-i] [-mname] [-nN] [-olist] [-raN] [-sN] [-Tdest] [-uN] [-z] [filename] ...

DESCRIPTION

troff formats text in the filenames for typesetting or laser printing. Input to troff is expected to consist of text interspersed with formatting requests and macros. If no filename argument is present, troff reads standard input. A minus sign (-) as a filename indicates that standard input should be read at that point in the list of input files.

The output of troff is usually piped through dpost(1) to create a printable postscript file (see EXAMPLES).

OPTIONS

The following options may appear in any order, but all must appear before the first filename.

-a Send an ASCII approximation of formatted output to standard output.

-f Do not print a trailer after the final page of output or cause the postprocessor to relinquish control of the device.

-Fdir
Search directory dir for font width or terminal tables instead of the system default directory.

-i Read standard input after all input files are exhausted.

-mname
Prepend the macro file /usr/share/lib/tmac/name to the input filenames. Note: most references to macro packages include the leading m as part of the name; for example, the man(5) macros reside in

/usr/share/lib/tmac/an. The macro directory can be changed by setting the TROFFMACS environment variable to a specific path. Be certain to include the trailing '/' (slash) at the end of the path.

-nN Number the first generated page N.

-olist
Print only pages whose page numbers appear in the comma-separated list of numbers and ranges. A range N-M means pages N through M; an initial -N means from the beginning to page N; and a final N- means from N to the end.

-q Quiet mode in nroff; ignored in troff.

-raN Set register a (one-character names only) to N.

-sN Stop the phototypesetter every N pages. On some devices, troff produces a trailer so you can change cassettes; resume by pressing the typesetter's start button.

-Tdest
Prepare output for typesetter dest. The following values can be supplied for dest:
post A PostScript printer; this is the default value.
aps Autologic APS-5.

-uN Set the emboldening factor for the font mounted in position 3 to N. If N is missing, then set the emboldening factor to 0.

-z Suppress formatted output. Only diagnostic messages and messages output using the .tm request are output.

OPERANDS
filename The file containing text to be processed by troff.

EXAMPLES

The following example shows how to print an input text file mytext, coded with formatting requests and macros. The input file contains equations and tables and must go through the tbl(1) and eqn(1) preprocessors before it is formatted by troff with ms macros, processed by dpost(1), and printed

by lp(1):

```
tbl mytext | eqn | troff -ms | dpost | lp
```

FILES

/tmp/trtmp	temporary file
/usr/share/lib/tmac/*	standard macro files
/usr/lib/font/*	font width tables for alternate mounted troff fonts
/usr/share/lib/nterm/*	terminal driving tables for nroff

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdoc

SEE ALSO

checknr(1), col(1), dpost(1), eqn(1), lp(1), man(1), nroff(1), tbl(1), attributes(5), man(5), me(5), ms(5)

NOTES

troff is not 8-bit clean because it is by design based on 7-bit ASCII.

uniq

uniq - report or filter out repeated lines in a file

SYNOPSIS

```
uniq [-c|-d|-u] [-f fields] [-s char] [input_file
     [output_file]]
uniq [-c|-d|-u] [-n] [+m]
     [input_file [output_file]]
```

DESCRIPTION

The uniq utility will read an input file comparing adjacent lines, and write one copy of each input line on the output. The second and succeeding copies of repeated adjacent input lines will not be written.

Repeated lines in the input will not be detected if they are not adjacent.

OPTIONS

The following options are supported:

-c
: Precede each output line with a count of the number of times the line occurred in the input.

-d
: Suppress the writing of lines that are not repeated in the input.

-f fields
: Ignore the first fields fields on each input line when doing comparisons, where fields is a positive decimal integer. A field is the maximal string matched by the basic regular expression:

  ```
  [[:blank:]]*[^[:blank:]]*
  ```

  If fields specifies more fields than appear on an input line, a null string will be used for comparison.

-s chars
: Ignore the first chars characters when doing comparisons, where chars is a positive decimal integer. If specified in conjunction with the

-f option, the first chars characters after the first fields fields will be ignored. If chars specifies more characters than remain on an input line, a null string will be used for comparison.

-u Suppress the writing of lines that are repeated in the input.

-n Equivalent to -f fields with fields set to n.
+m Equivalent to -s chars with chars set to m.

OPERANDS

The following operands are supported:

input_file A path name of the input file. If input_file is not specified, or if the input_file is -, the standard input will be used.

output_file A path name of the output file. If output_file is not specified, the standard output will be used. The results are unspecified if the file named by output_file is the file named by input_file.

EXAMPLES

The following example lists the contents of the uniq.test file and outputs a copy of the repeated lines.

```
example% cat uniq.test
This is a test.
This is a test.
TEST.
Computer.
TEST.
TEST.
Software.

example% uniq -d uniq.test
This is a test.
TEST.
example%
```

The next example outputs just those lines that are not repeated in the uniq.test file.

```
example% uniq -u uniq.test
TEST.
Computer.
Software.
example%
```

The last example outputs a report with each line preceded by a count of the number of times each line occurred in the file.

```
example% uniq -c uniq.test
   2 This is a test.
   1 TEST.
   1 Computer.
   2 TEST.
   1 Software.
example%
```

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of uniq: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWesu
CSI	Enabled

SEE ALSO

comm(1), pack(1), pcat(1), sort(1), uncompress(1), attributes(5), environ(5)

vi

vi, view, vedit - screen-oriented (visual) display editor based on ex

SYNOPSIS

```
/usr/bin/vi [ - | -s ] [-l] [-L] [-R] [ -r [ filename]]
     [ -t tag ] [-v] [-V] [-x] [ -wn ] [-C]
     [ +command | -c command ] filename...

/usr/bin/view [ - | -s ] [-l] [-L] [-R] [ -r [ filename]]
     [ -t tag ] [-v] [-V] [-x] [ -wn ] [-C]
     [ +command | -c command ] filename...

/usr/bin/vedit [ - | -s ] [-l] [-L] [-R] [ -r [ filename]]
     [ -t tag ] [-v] [-V] [-x] [ -wn ] [-C]
     [ +command | -c command ] filename...

/usr/xpg4/bin/vi [ - | -s ] [-l] [-L] [-R] [ -r [ filename]]
     [ -t tag ] [-v] [-V] [-x] [ -wn ]
     [-C] [ +command | -c command ] filename...

/usr/xpg4/bin/view [ - | -s ] [-l] [-L] [-R]
     [ -r [ filename]] [ -t tag ] [-v] [-V] [-x] [ -wn ]
     [-C] [ +command | -c command ] filename...

/usr/xpg4/bin/vedit [ - | -s ] [-l] [-L] [-R]
     [ -r [ filename]] [ -t tag ] [-v] [-V] [-x] [ -wn ]
     [-C] [ +command | -c command ] filename...
```

DESCRIPTION

vi (visual) is a display-oriented text editor based on an underlying line editor ex. It is possible to use the command mode of ex from within vi and to use the command mode of vi from within ex. The visual commands are described on this manual page; how to set options (like automatically numbering lines and automatically starting a new output line when you type carriage return) and all ex line editor commands are described on the ex(1) manual page.

When using vi, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

The view invocation is the same as vi except that the readonly flag is set.

The vedit invocation is intended for beginners. It is the same as vi except that the report flag is set to 1, the showmode and novice flags are set, and magic is turned off. These defaults make it easier to learn how to use vi.

OPTIONS

Invocation Options

The following invocation options are interpreted by vi (previously documented options are discussed in the NOTES section of this manual page):

- | -s — Suppress all interactive user feedback. This is useful when processing editor scripts.

-l — Set up for editing LISP programs.

-L — List the name of all files saved as the result of an editor or system crash.

-R — Readonly mode; the readonly flag is set, preventing accidental overwriting of the file.

-r filename — Edit filename after an editor or system crash. (Recovers the version of filename that was in the buffer when the crash occurred.)

-t tag — Edit the file containing the tag and position the editor at its definition.

-v — Start up in display editing state using vi. You can achieve the same effect by simply typing the -vi command itself.

-V — Verbose. When ex commands are read by means of standard input, the input will be echoed to standard error. This may be useful when processing ex commands within shell scripts.

-x — Encryption option; when used, vi simulates the X command of ex and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of the crypt command. The X command makes an educated guess to determine whether text read in is encrypted or

not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the -x option.

-wn Set the default window size to n. This is useful when using the editor over a slow speed line.

-C Encryption option; same as the -x option, except that vi simulates the C command of ex. The C command is like the X command of ex, except that all text read in is assumed to have been encrypted.

+command | -c command
Begin editing by executing the specified editor command (usually a search or positioning command).

/usr/xpg4/bin/vi
If both the -t tag and the -c command options are given, the -t tag will be processed first. That is, the file containing the tag is selected by -t and then the command is executed.

OPERANDS
The following operands are supported:

filename A file to be edited.

COMMAND SUMMARY

vi Modes

Command Normal and initial mode. Other modes return to command mode upon completion. ESC (escape) is used to cancel a partial command.

Input Entered by setting any of the following options: a A i I o O c C s S R. Arbitrary text may then be entered. Input mode is normally terminated with ESC character, or, abnormally, with an interrupt.

Last line Reading input for : / ? or !; terminate by typing a carriage return; an interrupt cancels termination.

Sample commands
In the descriptions, CR stands for carriage return and ESC

```
     stands for the escape key.

     <- | v | ^ ->    arrow keys move the cursor
     h j k l      same as arrow keys
     itextESC     insert text
     cwnewESC     change word to new
     easESC       pluralize word (end of word; append s;
                  escape from input state)
     x            delete a character
     dw           delete a word
     dd           delete a line
     3dd          delete 3 lines
     u            undo previous change
     ZZ           exit vi, saving changes
     :q!CR        quit, discarding changes
     /textCR      search for text
     ^U ^D        scroll up or down
     :cmdCR       any ex or ed command
  Counts before vi commands
     Numbers may be typed as a prefix to some commands.  They are
     interpreted in one of these ways.
     line/column number   z  G  |
     scroll amount        ^D  ^U
     repeat effect        most of the rest

  Interrupting, canceling
      ESC          end insert or incomplete cmd
      DEL          (delete or rubout) interrupts

  File manipulation
      ZZ              if file modified, write and exit; otherwise,
exit
      :wCR            write back changes
      :w!CR           forced write, if permission originally not
valid
      :qCR            quit
      :q!CR           quit, discard changes
      :e nameCR       edit file name
      :e!CR           reedit, discard changes
      :e + nameCR     edit, starting at end
      :e +nCR         edit starting at line n
      :e #CR          edit alternate file
      :e! #CR         edit alternate file, discard changes
      :w nameCR       write file name
      :w! nameCR      overwrite file name
      :shCR           run shell, then return
      :!cmdCR         run cmd, then return
```

```
      :nCR            edit next file in arglist
      :n argsCR       specify new arglist
      ^G              show current file and line
      :ta tagCR       position cursor to tag
```

In general, any ex or ed command (such as substitute or global) may be typed, preceded by a colon and followed by a carriage return.

Positioning within file

```
      ^F          forward screen
      ^B          backward screen
      ^D          scroll down half screen
      ^U          scroll up half screen
      nG          go to the beginning of the specified line (end
default),
                  where n is a line number
      /pat        next line matching pat
      ?pat        previous line matching pat
      n           repeat last / or ? command
      N           reverse last / or ? command
      /pat/+n     nth line after pat
      ?pat?-n     nth line before pat
      ]]          next section/function
      [[          previous section/function
      (           beginning of sentence
      )           end of sentence
      {           beginning of paragraph
      }           end of paragraph
      %           find matching ( ) { or }
```

Adjusting the screen

```
      ^L           clear and redraw window
      ^R           clear and redraw window if ^L is -> key
      zCR          redraw screen with current line at top of
window
      z-CR         redraw screen with current line at bottom of
window
      z.CR         redraw screen with current line at center of
window
      /pat/z-CR    move pat line to bottom of window
      zn.CR        use n-line window
      ^E           scroll window down 1 line
      ^Y           scroll window up 1 line
```

Marking and returning

```
      ``          move cursor to previous context
```

```
      ''           move cursor to first non-white space in line
      mx           mark current position with the ASCII lower-case
letter x
      `x           move cursor to mark x
      'x           move cursor to first non-white space in line
marked by x

  Line positioning
      H            top line on screen
      L            last line on screen
      M            middle line on screen
      +            next line, at first non-white
      -            previous line, at first non-white
      CR           return, same as +
      | v or j       next line, same column
      | ^ or k       previous line, same column

  Character positioning
      ^            first non white-space character
      0            beginning of line
      $            end of line
      l or ->      forward
      h or <-      backward
      ^H           same as <- (backspace)
      space        same as -> (space bar)
      fx           find next x
      Fx           find previous x
      tx           move to character prior to next x
      Tx           move to character following previous x
      ;            repeat last f, F, t, or T
      ,            repeat inverse of last f, F, t, or T
      n|           move to column n
      %            find matching ( { ) or }

  Words, sentences, paragraphs
      w            forward a word
      b            back a word
      e            end of word
      )            to next sentence
      }            to next paragraph
      (            back a sentence
      {            back a paragraph
      W            forward a blank-delimited word
      B            back a blank-delimited word
      E            end of a blank-delimited word

  Corrections during insert
```

```
      ^H            erase last character (backspace)
      ^W            erase last word
      erase         your erase character, same as ^H (backspace)
      kill          your kill character, erase this line of input
      \             quotes your erase and kill characters
      ESC           ends insertion, back to command mode
      CTRL-C        interrupt, suspends insert mode
      ^D            backtab one character; reset left margin of
autoindent
      ^^D           caret (^) followed by control-d (^D);
                    backtab to beginning of line;
                    do not reset left margin of autoindent
      0^D           backtab to beginning of line; reset left margin
of autoindent
      ^V            quote non-printable character

  Insert and replace
      a             append after cursor
      A             append at end of line
      i             insert before cursor
      I             insert before first non-blank
      o             open line below
      O             open above
      rx            replace single char with x
      RtextESC      replace characters

  Operators
     Operators are followed by a cursor motion,  and  affect  all
     text  that would have been moved over.  For example, since w
     moves over a word, dw deletes the word that would  be  moved
     over.   Double the operator, for example, dd to affect whole
     lines.
      d             delete
      c             change
      y             yank lines to buffer
      <             left shift
      >             right shift
      !             filter through command

  Miscellaneous Operations
      C             change rest of line (c$)
      D             delete rest of line (d$)
      s             substitute chars (cl)
      S             substitute lines (cc)
      J             join lines
      x             delete characters (dl)
      X             delete characters before cursor (dh)
```

```
     Y            yank lines (yy)
```

```
  Yank and Put
     Put inserts the text most recently deleted or  yanked;  how-
     ever,  if  a  buffer  is  named  (using the ASCII lower-case
     letters a - z), the text in that buffer is put instead.
      3yy          yank 3 lines
      3yl          yank 3 characters
      p            put back text after cursor
      P            put back text before cursor
      "xp          put from buffer x
      "xy          yank to buffer x
      "xd          delete into buffer x

  Undo, Redo, Retrieve
      u            undo last change
      U            restore current line
      .            repeat last change
      "dp          retrieve d'th last delete
```

USAGE

See largefile(5) for the description of the behavior of vi and view when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of vi: LC_CTYPE, LC_TIME, LC_MESSAGES, and NLSPATH.

FILES

/var/tmp	default directory where temporary work files are placed; it can be changed using the directory option (see the ex(1) set command)
/usr/share/lib/terminfo/?/*	compiled terminal description database
/usr/lib/.COREterm/?/*	subset of compiled terminal description database

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/vi
/usr/bin/view
/usr/bin/vedit

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Not enabled

/usr/xpg4/bin/vi
/usr/xpg4/bin/view
/usr/xpg4/bin/vedit

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4
CSI	Enabled

SEE ALSO

intro(1), ed(1), edit(1), ex(1), attributes(5), environ(5), largefile(5), standards(5)

Solaris Advanced User's Guide

AUTHOR

vi and ex were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

NOTES

Two options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard (see intro(1)). A -r option that is not followed with an option-argument has been replaced by -L and +command has been replaced by -c command.

The message file too large to recover with -r option, which is seen when a file is loaded, indicates that the file can be edited and saved successfully, but if the editing session is lost, recovery of the file with the -r option will not be possible.

The editing environment defaults to certain configuration options. When an editing session is initiated, vi attempts to read the EXINIT environment variable. If it exists, the editor uses the values defined in EXINIT, otherwise the values set in $HOME/.exrc are used. If $HOME/.exrc does not exist, the default values are used.

To use a copy of .exrc located in the current directory other than $HOME, set the exrc option in EXINIT or $HOME/.exrc. Options set in EXINIT can be turned off in a local .exrc only if exrc is set in EXINIT or $HOME/.exrc.

Tampering with entries in /usr/share/lib/terminfo/?/* or /usr/share/lib/terminfo/?/* (for example, changing or removing an entry) can affect programs such as vi that expect the entry to be present and correct. In particular, removing the "dumb" terminal may cause unexpected problems.

Software tabs using ^T work only immediately after the autoindent.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

The standard Solaris version of vi will be replaced by the POSIX.2-conforming version (see standards(5)) in the future. Scripts which use the ex family of addressing and features should use the /usr/xpg4/bin version of these utilities.

WC

wc - display a count of lines, words and characters in a file

SYNOPSIS

wc [-c | -m | -C] [-lw] [file...]

DESCRIPTION

The wc utility reads one or more input files and, by default, writes the number of newline characters, words and bytes contained in each input file to the standard output.

The utility also writes a total count for all named files, if more than one input file is specified.

wc considers a word to be a non-zero-length string of characters delimited by white space (for example, SPACE, TAB). See iswspace(3C) or isspace(3C).

OPTIONS

The following options are supported:

-c Count bytes.

-m Count characters.

-C Same as -m.

-l Count lines.

-w Count words delimited by white space characters or new line characters. Delimiting characters are Extended Unix Code (EUC) characters from any code set defined by iswspace().

If no option is specified the default is -lwc (count lines, words, and bytes.)

OPERANDS

The following operand is supported:

file A path name of an input file. If no file operands

are specified, the standard input will be used.

USAGE

See largefile(5) for the description of the behavior of wc when encountering files greater than or equal to 2 Gbyte (2**31 bytes).

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of wc: LC_CTYPE, LC_MESSAGES, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
CSI	Enabled

SEE ALSO

cksum(1), isspace(3C), iswalpha(3C), iswspace(3C), setlocale(3C), attributes(5), environ(5), largefile(5)

which

which - locate a command; display its pathname or alias

SYNOPSIS

which [filename] ...

DESCRIPTION

which takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are taken from the user's .cshrc file.

FILES

~/.cshrc source of aliases and path values
/usr/bin/which

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

SEE ALSO

csh(1), attributes(5)

DIAGNOSTICS

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

NOTES

which is not a shell built-in command; it is the UNIX command, /usr/bin/which

BUGS

Only aliases and paths from ~/.cshrc are used; importing from the current environment is not attempted. Must be executed by csh(1), since only csh knows about aliases.

To compensate for ~/.cshrc files in which aliases depend upon the prompt variable being set, which sets this variable to NULL. If the ~/.cshrc produces output or prompts for input when prompt is set, which may produce some strange results.

who

who - who is on the system

SYNOPSIS

```
/usr/bin/who [ -abdHlmpqrstTu ] [ file ]
/usr/bin/who -q [ -n x ] [ file ]
/usr/bin/who am i
/usr/bin/who am I

/usr/xpg4/bin/who [ -abdHlmpqrtTu ] [ file ]
/usr/xpg4/bin/who -q [ -n x ] [ file ]
/usr/xpg4/bin/who -s [ -bdHlmpqrtu ] [ file ]
/usr/xpg4/bin/who am i
/usr/xpg4/bin/who am I
```

DESCRIPTION

The who utility can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX system user. It examines the /var/adm/utmp file to obtain its information. If file is given, that file (which must be in utmp(4) format) is examined. Usually, file will be /var/adm/wtmp, which contains a history of all the logins since the file was last created.

The general format for output is:

```
name [state] line time [idle] [pid] [comment] [exit]
```

where:

name	user's login name.
state	capability of writing to the terminal.
line	name of the line found in /dev.
time	time since user's login.
idle	time elapsed since the user's last activity.
pid	user's process id.
comment	comment line in inittab(4).
exit	exit status for dead processes.

OPTIONS

The following options are supported:

-a Process /var/adm/utmp or the named file with -b, -d, -l, -p, -r, -t, -T, and -u options turned on.

-b Indicate the time and date of the last reboot.

-d Display all processes that have expired and not been respawned by init. The exit field appears for dead processes and contains the termination and exit values (as returned by wait(3B)), of the dead process. This can be useful in determining why a process terminated.

-H Output column headings above the regular output.

-l List only those lines on which the system is waiting for someone to login. The name field is LOGIN in such cases. Other fields are the same as for user entries except that the state field does not exist.

-m Output only information about the current terminal.

-n x Take a numeric argument, x, which specifies the number of users to display per line. x must be at least 1. The -n option may only be used with -q.

-p List any other process which is currently active and has been previously spawned by init. The name field is the name of the program executed by init as found in /sbin/inittab. The state, line, and idle fields have no meaning. The comment field shows the id field of the line from /sbin/inittab that spawned this process. See inittab(4).

-q (quick who) display only the names and the number of users currently logged on. When this option is used, all other options are ignored.

-r Indicate the current run-level of the init process.

-s (default) List only the name, line, and time fields.

/usr/bin/who

-T Same as the -s option, except that the state idle, pid, and comment, fields are also written. state is one of the following characters:

+ The terminal allows write access to other users.
- The terminal denies write access to other users.
? The terminal write-access state cannot be determined.

/usr/xpg4/bin/who

-T Same as the -s option, except that the state field is also written. state is one of the characters listed under the /usr/bin/who version of this option.

If the -u option is used with -T, the idle time is added to the end of the previous format.

-t Indicate the last change to the system clock (using the date utility) by root. See su(1M) and date(1).

-u List only those users who are currently logged in. The name is the user's login name. The line is the name of the line as found in the directory /dev. The time is the time that the user logged in. The idle column contains the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore ``current''. If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The pid is the process-ID of the user's shell. The comment is the comment field associated with this line as found in /sbin/inittab (see inittab(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, and so forth.

OPERANDS

The following operands are supported:

am i
am I In the "C" locale, limit the output to describing the invoking user, equivalent to the -m option. The am and i or I must be separate arguments.

file Specify a path name of a file to substitute for the database of logged-on users that who uses by default.

ENVIRONMENT

See environ(5) for descriptions of the following environment variables that affect the execution of who: LC_CTYPE, LC_MESSAGES, LC_TIME, and NLSPATH.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

FILES

/sbin/inittab	script for init.
/var/adm/utmp	current user and accounting information
/var/adm/wtmp	historic user and accounting information

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

/usr/bin/who

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

/usr/xpg4/bin/who

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxcu4

SEE ALSO

date(1), login(1), mesg(1), init(1M), su(1M), wait(3B), inittab(4), utmp(4), attributes(5), environ(5), xpg4(5)

NOTES

Super-user: After a shutdown to the single-user state, who returns a prompt; since /var/adm/utmp is updated at login time and there is no login in single-user state, who cannot report accurately on this state. who am i, however, returns the correct information.