System V Interface Definition, Fourth Edition
Volume 2

Novell, Inc.
122 East 1700 South
Provo, UT 84606
U.S.A.

IMPORTANT NOTE TO USERS

While every effort has been made to ensure the accuracy of all information in this document, Novell assumes no liability to any party for any loss of damage caused by errors or omissions or by statements of any kind in the *System V Interface Definition*, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accident, or any other cause. Novell further assumes no liability arising out of the application or use of any product or system described herein; nor any liability for incidental or consequential damages arising from the use of this document. Novell disclaims all warranties regarding the information contained herein, whether expressed, implied or statutory, including implied warranties of merchantability or fitness for a particular purpose.

Novell makes no representation that the interconnection of products in the manner described herein will not infringe on existing or future patent rights, nor do the descriptions contained herein imply the granting or license to make, use or sell equipment constructed in accordance with this description.

Novell reserves the right to make changes without further notice to any products herein to improve reliability, function, or design.

TRADEMARKS

Ann Arbor is a trademark of Ann Arbor Terminals, Inc.
Beehive is a trademark of Beehive International.
Concept is a trademark of Human Designed Systems, Inc.
HP is a trademark of Hewlett–Packard Co.
LSI is a trademark of Lear Siegler, Inc.
Micro–Term, ACT and MIME are trademarks of Micro–Term, Inc.
OSF/Motif is a trademark of the Open Software Foundation
PostScript is a trademark of Adobe Systems.
Tektronix and Tektronix 4010 are registered trademarks of Tektronix, Inc.
TeleVideo is a registered trademark of TeleVideo Systems, Inc.
Teleray is a trademark of Research, Inc.
Teletype is a registered trademark of AT&T.
The X Window System is a trademark of MIT.
UNIX is a registered trademark in the USA and other countries, licensed exclusively through X/Open Company, Ltd.
VT100 is a trademark of Digital Equipment Corporation.
X/Open is a trademark of X/Open Company Limited.

Volume 2 Table of Contents

Basic Utilities Introduction

The Basic Utilities Extension defines an environment that provides basic user-level functionality. It includes the **sh** (shell) command interpreter, shell programming aids, facilities for basic directory and file manipulation, and facilities for text file editing and processing.

The Base System is prerequisite for support of the Basic Utilities Extension.

Summary of Commands and Utilities

The following commands and utilities are supported by the Basic Utilities Extension. Items marked with a (*) are Level 2, as defined in the *General Introduction* to this volume. Items marked with a (‡) are internationalized and may reference environment variables for localization information. [See envvar(BA_ENV)]. Items marked with a (†) are new to this issue of the SVID.

ar‡	**defadm**‡	**kill**‡	**pfmt** *‡	**strchg**
awk‡	**df**‡	**lfmt** *‡	**pg***‡	**strconf**
banner‡	**diff3**	**line**	**pr**‡	**sum**‡
basename‡	**diff**‡	**listusers**	**printf**	**tail**‡
cal	**dirname**‡	**ln**‡	**ps**‡	**tee**‡
calendar	**du**‡	**ls**‡	**pwd**‡	**test**‡
cat‡	**echo**	**mail**‡	**red**‡	**touch**‡
cd‡	**ed**‡	**mkdir**‡	**rm**‡	**tr**‡
chmod‡	**expr**‡	**more**	**rmail**‡	**true**
cmp‡	**false**	**mv**‡	**rmdir**‡	**umask**‡
col‡	**file**‡	**nawk**‡	**rsh**‡	**uname**‡
comm‡	**find**‡	**nl**‡	**sed**‡	**uncompress**
compress‡	**fmt**	**nohup**‡	**sh**‡	**uniq**‡
cp‡	**gettxt**‡	**pack**‡	**sleep**‡	**unpack**‡
cpio‡	**grep**‡	**page**	**sort**‡	**wait**‡
ctags‡	**head**	**paste**‡	**spell**‡	**wc**‡
cut‡	**iconv**‡	**pcat**‡	**split**‡	**zcat**
date‡	**jsh**‡			

Organization of Technical Information

The ‘‘Basic Commands and Utilities’’ chapter provides manual page descriptions of commands and utilities supported by this extension.

Basic Commands And Utilities

The following section contains the manual pages for the BU_CMD routines.

NAME

ar – maintain portable archive or library

SYNOPSIS

ar [-V] -*key* **[***arg***] [***posname***]** *afile* **[***name* **. . .]**

DESCRIPTION

The **ar** command maintains groups of files combined into a single archive file. Its main use is to create and update library files. However, it can be used for any similar purpose. If an archive is composed of printable files, the entire archive is printable.

When **ar** creates an archive, it creates headers in a format that is portable across all machines. The archive symbol table is used by the link editor **ld** to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by **ar** when there is at least one object file in the archive. The archive symbol table is in a specially named file that is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the **ar** command is used to create or update the contents of such an archive, the symbol table is rebuilt. The **s** argument to *key*, described below, will force the symbol table to be rebuilt.

-V Cause **ar** to print its version number on standard error.

-*key* **[***arg***]**

key is formed with one of the following characters: **drqtpmx**. *arg* is formed with one of more of the following letters: **vucls**. An additional single-character argument to *key*, called the positioning character (chosen from one of the following letters: **abi**), can be used with *key* characters **r** and **m**. *key* characters are described below.

posname Archive member name used as a reference point in positioning other files in the archive.

afile Archive file.

name One or more constituent files in the archive file.

The meanings of the *key* characters are as follows:

-d Delete the named files from the archive file.

-m Move the named files to the end of the archive. If an optional positioning character from the set **abi** is used, the *posname* argument must be present and specifies that new files are to be placed after **a** or before **b** or **i** *posname*. Otherwise new files are placed at the end.

-p Print the named files in the archive.

-q

new files are to be placed after **a** or before **b** or **i** *posname*. Otherwise new files are placed at the end.

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

-u Update older files. When used with the **-r** option, files within the archive are replaced only if the corresponding *file* has a modification time that is at least as new as the modification time of the file within the archive.

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

The meanings of the other key arguments are as follows:

v Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with the **-t** option, give a long listing of all information about the files. When used with **x**, **d**, or **r**, print the name of the file preceding each extraction. When used with **p**, write the file to standard output.

c Suppress the message that is produced by default when *afile* is created.

s Force the regeneration of the archive symbol table even if **ar** is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the **strip**(SD_CMD) command has been used on the archive.

SEE ALSO

ld (SD_CMD), **lorder**(SD_CMD), **strip**(SD_CMD)

LEVEL

Level 1.

''1'' files

NAME

awk – pattern-directed scanning and processing language

SYNOPSIS

`awk` [`-F` *fs*] [*prog*] [*inputfile* ...]

`awk` [`-F` *fs*] [`-f` *progfile*] [*inputfile* ...]

DESCRIPTION

`awk` scans each input *inputfile* for lines that match any of a set of patterns specified literally in *prog* or in a file specified as `-f` *progfile*. With each pattern there can be an associated action that will be performed when a line of a *inputfile* matches the pattern. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern. The inputfile name `-` means the standard input. Any *inputfile* of the form *var=value* is treated as an assignment, not a filename.

An input line is made up of fields separated by white space, or by the regular expression assigned to special variable `FS`. The `-F` *fs* option defines the input field separator as the regular expression *fs*.

The fields are denoted `$1`, `$2`, ...; `$0` refers to the entire line.

A pattern-action statement has the form

```
pattern { action }
```

A missing `{ action }` means print the line; a missing pattern always matches. Pattern-action statements are separated by newlines or semicolons.

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

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

Statements are terminated by semicolons, newlines or right braces. An empty *expression-list* stands for `$0`. String constants are quoted `" "`, with the usual C language escapes recognized within. Expressions take on string or numeric values as appropriate, and are built using the operators `+ - * / % ^` (exponentiation), and concatenation (indicated by a blank). The operators `! ++ -- += -= *= /= %= ^= > >= < <= == != ?:` are also available in expressions. Variables may be scalars, array elements (denoted *x*`[`*i*`]`) or fields. Variables are initialized to the null

string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. Multiple subscripts such as `[i,j,k]` are permitted; the constituents are concatenated, separated by the value of SUBSEP.

The `print` statement prints its arguments on the standard output (or on a file if *>file* or *>>file* is present or on a pipe if *|cmd* is present), separated by the current output field separator, and terminated by the output record separator. *file* and *cmd* may be literal names or parenthesized expressions; identical string values in different statements denote the same open file. The `printf` statement formats its expression list according to the format [see printf(BA_LIB)]. The built-in function `close(` *expr*) closes the file or pipe *expr*.

The customary functions `exp`, `log`, `sqrt`, `sin`, `cos`, and `atan2` are built in. Other built-in functions are:

`length`
: the length in characters of its argument taken as a string, or of `$0` if no argument.

`rand` random number on (0,1)

`srand`
: sets the seed for `rand`

`int` truncates to an integer value

`substr(`*s*`, `*m*`, `*n*`)`
: the *n*-character substring of *s* that begins at position *m* counted from 1.

`index(`*s*`, `*t*`)`
: the position in *s* where the string *t* occurs, or 0 if it does not.

`match(`*s*`, `*r*`)`
: the position in *s* where the regular expression *r* occurs, or 0 if it does not. The variables `RSTART` and `RLENGTH` are set to the position and length of the matched string.

`split(`*s*`, `*a*`, `*fs*`)`
: splits 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.

`sub(`*r*`, `*t*`, `*s*`)`
: substitutes *t* for the first occurrence of the regular expression *r* in the string *s*. If *s* is not given, `$0` is used. `sub` returns the number of replacements.

`gsub` same as `sub` except that all occurrences of the regular expression are replaced; `gsub` returns the number of replacements.

`sprintf(`*fmt*`, `*expr*`, ...)`
: the string resulting from formatting *expr* ... according to the `printf` format *fmt*

`system(`*cmd*`)`
: executes *cmd* and returns its exit status

The ''function'' `getline` sets `$0` to the next input record from the current input file; `getline` <*file* sets `$0` to the next record from *file*. `getline` *x* sets variable *x* instead. Finally, `cmd|getline` pipes the output of *cmd* into `getline`; each call of `getline` returns the next line of output from *cmd*. In all cases, `getline` returns `1` for a successful input, `0` for end of file, and `-1` for an error.

Patterns are arbitrary Boolean combinations (with `! || && )` of regular expressions and relational expressions. Regular expressions are as in `egrep` [see egrep(AU_CMD)]. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions, using the operators `~` and `!~`. */re/* is a constant regular expression; any string (constant or variable) may be used as a regular expression, except in the position of an isolated regular expression in a pattern.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines from an occurrence of the first pattern though an occurrence of the second.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression
expression in array-name
(expr,expr,...) in array-name

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

The special patterns `BEGIN` and `END` may be used to capture control before the first input line is read and after the last. `BEGIN` and `END` do not combine with other patterns.

Variable names with special meanings:

`FS`	regular expression used to separate fields; also settable by option `-F` *fs*.
`NF`	number of fields in the current record.
`NR`	ordinal number of the current record.
`FNR`	ordinal number of the current record in the current file.
`FILENAME`	the name of the current input file.
`RS`	input record separator (default newline).
`OFS`	output field separator (default blank).
`ORS`	output record separator (default newline).
`OFMT`	output format for numbers (default **%.6g**).
`SUBSEP`	separates multiple subscripts (default 034).
`ARGC`	argument count, assignable.

ARGV argument array, assignable; non-null members are taken as filenames.

Functions may be defined (at the position of a pattern-action statement) thus:

```
function foo(a, b, c) { ...; return x }
```

Parameters are passed by value if scalar and by reference if array name; functions may be called recursively. Parameters are local to the function; all other variables are global.

EXAMPLE

```
length > 72
```
Print lines longer than 72 characters.

```
{ print $2, $1 }
```
Print first two fields in opposite order.

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```
Same, with input fields separated by comma and/or blanks and tabs.

```
     { s += $1 }
END  { print "sum is", s, " average is", s/NR }
```
Add up first column, print sum and average.

```
/start/, /stop/
```
Print all lines between start/stop pairs.

```
BEGIN     {    # Simulate echo(1)
     for (i = 1; i < ARGC; i++) printf "%s ", ARGV[i]
     printf "\n"
     exit }
```

SEE ALSO

egrep(AU_CMD), lex(SD_CMD), printf(BA_LIB), sed(BU_CMD).

LEVEL

Level 1.

NAME

banner – make large letters

SYNOPSIS

`banner` *strings*

DESCRIPTION

The command `banner` prints each argument in large letters (across the page) on the standard output, putting each argument on a separate line. Spaces can be included in an argument by surrounding it with quotes. The maximum number of characters that can be accommodated in a line is implementation dependent; excess characters are simply ignored.

ERRORS

Non-ASCII characters specified in *strings* may not be output correctly.

SEE ALSO

echo(BU_CMD).

LEVEL

Level 1.

NAME

basename, **dirname** – deliver portions of path names

SYNOPSIS

basename *string* **[***suffix***]**

dirname *string*

DESCRIPTION

basename deletes any prefix ending in **/** and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside substitution marks (**` `**) within shell procedures. The *suffix* is a pattern as defined on the **ed**(BU_CMD) manual page.

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

The **LC_CTYPE** environment variable defines the codesets used in the pathname [see **LANG** on **envvar**(BA_ENV)].

EXAMPLES

The following example, invoked with the argument **/home/sms/personal/mail** sets the environment variable **NAME** to the file named **mail** and the environment variable **MYMAILPATH** to the string **/home/sms/personal**.

```
NAME=`basename $HOME/personal/mail`
MYMAILPATH=`dirname $HOME/personal/mail`
```

This shell procedure, invoked with the argument **/usr/src/bin/cat.c**, compiles the named file and moves the output to **cat** in the current directory:

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

SEE ALSO

sh(BU_CMD)

LEVEL

Level 1.

NAME

cal – print calendar

SYNOPSIS

cal [[*month***]** *year***]**

DESCRIPTION

cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present month is printed. The *month* is a number between 1 and 12. The *year* can be between 1 and 9999. The calendar produced is in the form of a Gregorian calendar (as used in Western Europe and the United States), but the month and the abbreviated day names are taken from the locale given by the environment variable **LC_TIME**.

cal examines the environment variables **LC_TIME** to determine the names of the months and days, and **LC_CTYPE** for how to print the characters the names are composed from. If the abbreviated day name in the locale entry is longer than two screen columns in width, it is truncated to two columns.

LEVEL

Level 1.

NOTICES

The command **cal 83** refers to the year 83, not 1983.

NAME

calendar – reminder service

SYNOPSIS

calendar

DESCRIPTION

calendar consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as **Aug. 24**, **august 24**, **8/24**, and so on, are recognized, but not **24 August** or **24/8**. On weekends "tomorrow" extends through Monday. **calendar** can be invoked regularly by using the **crontab**(AU_CMD) or **at**(AU_CMD) commands.

If the environment variable **DATEMSK** is set, **calendar** will use its value as the full path name of a template file containing format strings. The strings consist of field descriptors and text characters and are used to provide a richer set of allowable date formats in different languages by appropriate settings of the environment variable **LC_ALL**, or **LC_TIME**. variables **LC_ALL**, **LC_TIME**, and **LANG** [see **envvar**(BA_ENV)]. The **LC_CTYPE** environment variable is also examined for details of the codesets used in the format strings. [See **date**(BU_CMD) for the allowable list of field descriptors.]

EXAMPLES

The following example shows the possible contents of a template:

```
%B %eth of the year %Y
```

%B represents the full month name, **%e** the day of month and **%Y** the year (4 digits).

If **DATEMSK** is set to this template, the following **calendar** file would be valid:

```
March 7th of the year 1989 <Reminder>
```

SEE ALSO

at(AU_CMD), **cron**(AU_CMD), **crontab**(AU_CMD), **date**(BU_CMD), **mail**(BU_CMD)

LEVEL

Level 1.

NAME

cat – concatenate and print files

SYNOPSIS

cat [-suv] [*file* **. . .]**

DESCRIPTION

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

```
cat file
```

prints the contents of **file** on your terminal, and

```
cat file1 file2 >file3
```

concatenates **file1** and **file2**, and writes the results in **file3**. If no input file is given, or if the argument **-** is encountered, **cat** reads from the standard input. **cat** processes supplementary code set characters according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar**(BA_ENV)].

The following options apply to **cat**:

-u The output is not buffered. (The default is buffered output.)

-s **cat** is silent about non-existent files.

-v Causes non-printing characters (with the exception of tabs, new-lines, and form-feeds) to be printed visibly. ASCII control characters (octal 000 – 037) are printed as **^** *n*, where *n* is the corresponding ASCII character in the range octal 100 – 137 (@, A, B, C, . . ., X, Y, Z, [, \,], ^, and _); the DEL character (octal 0177) is printed **^?**. Other non-printable characters are printed as **M-** *x*, where *x* is the ASCII character specified by the low-order seven bits. All supplementary code set characters are considered to be printable.

Errors

cat returns the following values:

0 If all input files were output successfully.
>0 If an error occurred while accessing one or more input files.

SEE ALSO

cp(BU_CMD), **pg**(BU_CMD), **pr**(BU_CMD)

LEVEL

Level 1.

NAME

cd – change working directory

SYNOPSIS

cd [*directory***]**

DESCRIPTION

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

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

SEE ALSO

chdir(BA_OS), **pwd**(BU_CMD), **sh**(BU_CMD)

LEVEL

Level 1.

NAME

chmod – change file mode

SYNOPSIS

chmod [-R] *mode file* **. . .**

chmod [-R][ugoa]{ + | - | = }[rwxlstugo] *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.

An absolute *mode* is specified using octal numbers:

chmod *nnnn file* **. . .**

where *n* is 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**. This bit is ignored if the file is a directory; it may be set or cleared only using the symbolic mode.
1000	Turn on sticky bit [see **chmod**(BA_OS)].
0400	Allow read by owner.
0200	Allow write by owner.
0100	Allow execute (search in directory) by owner.
0070	Allow read, write, and execute (search) by group.
0007	Allow read, write, and execute (search) by others.

On execution, the **setuid** and **setgid** modes affect interpreter scripts only if the first line of those scripts is

#! *pathname* **[***arg***]**

where *pathname* is the path of a command interpreter, such as **sh**. [See **exec**(BA_OS).]

A symbolic *mode* is specified in the following format:

chmod [*who* **]** *operator* **[***permission(s)***]** *file* **. . .**

who is 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**.

operator is one of +, –, or =, signifying how permissions are to be changed:

+	Add permissions.

-	Take away permissions.
=	Assign permissions absolutely.

Unlike other symbolic operations, **=** has an absolute effect in that it resets all other bits. Omitting *permission*(s) is useful only with **=** to take away all permissions.

permission(s) is any compatible combination of the following letters:

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

The **X** represents the execute permission of a file only if the file is a directory, or if the current (unmodified) file permissions have at least one execute bit present. If neither of these conditions are true, it will be ignored.

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.

Multiple symbolic modes separated by commas may be given, though no spaces may intervene between these modes. Operations are performed in the order given. Multiple symbolic letters following a single operator cause the corresponding operations to be performed simultaneously.

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. When locking is requested, the group ID of the user must be the same as the group ID of the file. 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 a privileged user) may change that file's or directory's mode. Only a privileged user may set the sticky bit on a non-directory file. Otherwise, **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.

The **-R** option recursively descends through directory arguments, setting the mode for each file as described above. If a symbolic link is encountered whose target is a directory, the permission of the directory is changed. That directory's contents are *not* recursively traversed.

USAGE

Deny execute permission to everyone:

chmod a-x *file*

Allow read permission to everyone:

chmod 444 *file*

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

chmod go+rw *file*
chmod 066 *file*

Cause a file to be locked during access:

chmod +l *file*

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

chmod =rwx,g+s *file*
chmod 2777 *file*

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

SEE ALSO

chmod(BA_OS), **getacl**(ES_CMD), **ls**(BU_CMD)

LEVEL

Level 1. The octal mode format of *mode* is Level 2.

NOTICES

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

Normally, the effective user and group ID of a process is the user and group ID of the invoking process. If the set-user-ID (set-group-ID) on execution mode bit of an executable file is set, the effective user (group) ID of the process, when the file is invoked, is the owner (group) ID of the executable file. The real user ID and real group ID of the new process remain the same as those of the calling process.

Setting the ''set-group-ID on execution'' bit on a directory (via the **g+s** option) means that any files subsequently created in that directory will automatically be given the group ID of that directory.

Neither set-user-ID nor set-group-ID mode bits affect shell script privileges.

When symbolic links are created by **ln**, they are made with permissions set to read, write, and execute for owner, group, and world (**777**). A **chmod** applied to a symbolic link acts on the target of the link, not on the link itself.

The symbolic modes should be used in preference to the octal representation, since the octal representation may not be supported in future releases.

If *who* is not specified, POSIX.2 requires use of **umask**. Use the **POSIX2** environmental variable to get POSIX.2 behavior. The POSIX.2 behavior is inconsistent with existing System V behavior.

NAME

cmp – compare two files

SYNOPSIS

cmp [**-l**] [**-s**] *file1 file2* [*skip1* [*skip2*]]

DESCRIPTION

The two files are compared. (If *file2* is **-**, the standard input is used.) Under default options, **cmp** makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted. *skip1* and *skip2* are initial byte offsets into *file1* and *file2* respectively, and may be either octal or decimal; a leading respectively, and may be either octal or decimal; the form of the number is determined by the environment variable **LC_NUMERIC** (in the **C** locale, a leading **0** denotes an octal number). [see **LANG** on **envvar**(BA_ENV)]. **0** denotes octal.

Options

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

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

Errors

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

SEE ALSO

comm (BU_CMD), **diff** (BU_CMD)

LEVEL

Level 1.

NAME

col – filter reverse line-feeds

SYNOPSIS

`col [-bfpx]`

DESCRIPTION

The command `col` reads from the standard input and writes to the standard output. It performs the line overlays implied by reverse line feeds, and by forward and reverse half-line feeds.

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

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

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

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

On input, the only control characters accepted are space, backspace, tab, return, newline, `SI`, `SO`, `VT`, reverse line feed, forward half-line feed, and reverse half-line feed. The `VT` character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

The `ASCII` codes for the control functions and line-motion sequences mentioned above are given in the table below. `ESC` stands for the ASCII "escape" character, with the octal code 033; `ESC`-*x* means a sequence of two characters, `ESC` followed by the character *x*.

reverse line feed	ESC-7
reverse half-line feed	ESC-8
forward half-line feed	ESC-9
vertical tab (VT)	013
start-of-text (SO)	016
end-of-text (SI)	017

Normally, `col` will remove any escape sequences found in its input that are unknown to it; the `-p` option may be used to force these to be passed through unchanged. The use of this option is discouraged unless the user is aware of the consequences.

USAGE

General.

Local vertical motions that would result in backing up past the first line of the document are ignored. As a result, the first line must not have any superscripts.

LEVEL

Level 1.

NAME

comm – select or reject lines common to two sorted files

SYNOPSIS

comm [-123] *file file2*

DESCRIPTION

comm reads *file1* and *file2*, which should be ordered in the current locale's collating sequence [see **sort**(BU_CMD)], and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name **-** means the standard input.

Flags **1**, **2**, or **3** suppress printing of the corresponding column. Thus **comm -12** prints only the lines common to the two files; **comm -23** prints only lines in the first file but not in the second; **comm -123** prints nothing.

The lines are compared using the current locale's collation sequence, set by the environment variable **LC_COLLATE**. [see **LANG** in **envvar**(BA_ENV) for the locale environment variables, and their effects on collation.] Note that if the files were **sort**ed with a different collation sequence from each other, or from that under which **comm** is executed, the results will be meaningless.

The **LC_CTYPE** environment variable determines the codesets used in the command line arguments and the files. [see **LANG** on **envvar**(BA_ENV)]. Note that if this variable differs when the files are sorted, or when **comm** is executed, the output of **comm** will be meaningless. Note also, that if **LC_CTYPE** and **LC_COLLATE** are set to different values, meaningful results cannot be guaranteed.

SEE ALSO

cmp (BU_CMD), **diff** (BU_CMD), **join** (AU_CMD), **sort** (BU_CMD), **uniq** (BU_CMD)

LEVEL

Level 1.

NAME

compress, **uncompress**, **zcat** – compress data for storage, uncompress and display compressed files

SYNOPSIS

compress [-fcv] [-b *bits***]** *file...*

uncompress [-c] *file...*

zcat *file...*

DESCRIPTION

compress takes a file and compresses it to the smallest possible size, creates a compressed output file, and removes the original file unless the **-c** option is present. Compression is achieved by encoding common strings within the file. **uncompress** restores a previously compressed file to its uncompressed state and removes the compressed version. **zcat** uncompresses and displays a file on the standard output.

If no file is specified on the command line, input is taken from the standard input and the output is directed to the standard output. Output defaults to a file with the same filename as the input file with the suffix **.Z** or it can be directed through the standard output. The output files have the same permissions and ownership as the corresponding input files or the user's standard permissions if output is directed through the standard output.

If no space is saved by compression, the output file is not written unless the **-F** flag is present on the command line.

Options

The following options are available from the command line:

-b *bits* Specifies the maximum number of bits to use in encoding.

–c Writes output on the standard output and does not remove original file.

-f Forces output file to be written, even if one already exists, and even if no space is saved by compressing.

-v Prints the name of the file being compressed and the percentage of compression achieved. With **uncompress**, the name of the uncompressed file is printed.

SEE ALSO

ar (BU_CMD), **cat** (BU_CMD), **pack** (BU_CMD), **tar** (AU_CMD)

LEVEL

Level 1.

NAME

cp – copy files

SYNOPSIS

cp [-r] [-i] [-e *extent_opt***]** *file1* **[***file2* **. . .]** *target*

DESCRIPTION

The **cp** command copies *filen* to *target*. *filen* and *target* may not have the same name. [Care must be taken when using **sh**(BU_CMD) metacharacters.] If *target* is not a directory, only one file may be specified before it; if it is a directory, more than one file may be specified. If *target* does not exist, **cp** creates a file named *target*. If *target* exists and is not a directory, its contents are overwritten. If *target* is a directory, the file(s) are copied to that directory.

The following options are recognized:

-i **cp** will prompt for confirmation whenever the copy would overwrite an existing *target*. An affirmative response means that the copy should proceed [the affirmative response is locale dependent: **y** in the **C** locale, see **LANG** on **envvar**(BA_ENV)]. Any other answer prevents **cp** from overwriting *target*. The **-i** option remains in effect even if the standard input is not a terminal.

-f **cp** will attempt to overwrite an existing target. If a file descriptor for target cannot be obtained, **cp** will attempt to unlink target and proceed. See NOTICES below.

-r (if *filen* is a directory) Copy the directory and all its files, including any subdirectories and their files. (If it exists, *target* must be a directory.) **-r** is multithreaded and uses the enhanced **nftw** (walk a file tree). See NOTICES below.

-R Copy a file hierarchy in the same fashion as **-r**. However, instead of copying special files (device files, FIFOs, or symbolic links) by copying their contents, create a target file with the same file type as *filen*. Normal files are copied in the same fashion as for **-r**. **-R** is multithreaded and uses the enhanced **nftw** (walk a file tree). See NOTICES below.

-e *extent_opt*

Specify how to handle a file that has extent attribute information. Extent attributes could include reserved space, a fixed extent size, and extent alignment. It may not be possible to preserve the information if the destination file system does not support extent attributes, has a different block size than the source file system, or lacks free extents appropriate to satisfy the extent attribute requirements. Valid values for *extent_opt* are:

warn Issue a warning message if extent attribute information cannot be kept (default).

force Fail the copy if extent attribute information cannot be kept.

ignore Ignore extent attribute information entirely.

If *filen* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file.

SEE ALSO

chmod(BU_CMD), **cpio**(BU_CMD), **rm**(BU_CMD)

LEVEL

Level 1.

NOTICES

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.

cp without the **-R** options hangs if *file* is a pipe.

It is not considered an error if more than one of the **-f** or **-i** options are specified. The last option specified will determine **cp**'s behavior.

The algorithm used to efficiently distribute the tree walking among various threads may affect the order in which files and directories are copied. This order may not match the hierarchy of the original input tree. The result, however, will match the source. Issuing a BREAK to the command while is is executing will yield a partially completed tree where files and directories may appear to have been copied arbitrarily. No particular order is guaranteed. Because they are not dependent on any particular order for populating the tree, **cp -r** and **cp -R** execute much more quickly than previous versions.

NAME

cpio – copy file archives in and out

SYNOPSIS

`cpio -o[acBLv] [-C` *size*`] [-H` *header*`]`

`cpio -i[Bcdkmrtuvf] [-C` *size*`] [-H` *header*`] [-R` *id*`] [`*patterns*`]`

`cpio -p[adlLmuv] [-R` *id*`]` *directory*

DESCRIPTION

The command `cpio -o` (copy out) reads the standard input to obtain a list of pathnames and copies those files onto the standard output in archive form, including pathname and status information.

The command `cpio -i` (copy in) extracts files from the standard input, which is assumed to be the product of a previous `cpio -o`. Only files with names that match *patterns* are selected. The arguments *patterns* are simple regular expressions given in the name-generating notation of the shell [see sh(BU_CMD)]. In *patterns*, meta-characters `?`, `*`, and `[...]` match the `/` character and a backslash (`\`) is used as an escape character within the pattern. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is `*` (i.e., select all files). When `cpio` is invoked from the shell, each *pattern* should be quoted; otherwise the shell may expand the *pattern* to the names of files in the current directory. The extracted files are conditionally created and copied into the current directory tree based on the options described below. The permissions of the files are those of the previous `cpio -o.` `cpio -i` is affected by the `umask` for non-priviledged users. The owner and group of the files are those of the current user unless the user has appropriate privileges, which causes `cpio` to retain the owner and group of the files of the previous `cpio -o.`

The command `cpio -p` (pass) copies in and out in a single operation. Destination pathnames are interpreted relative to the destination *directory*.

The meanings of the available options are:

`-a` Reset access times of input files after they have been copied. [When option `-l` (see below) is also specified, access times of the linked files are not reset.]

`-B` Block input/output 5120 bytes to the record. (This does not apply to the `-p` option; it is meaningful only with data directed to or from character special files.)

`-d` Creates directories as needed.

`-c` Write header information in ASCII character form for portability.

`-r` Rename files interactively. If the user types a null line, the file is skipped. If the user types a period, the original pathname is retained.

`-t` Print a table of contents of the input. No files are created.

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

`-v` Verbose: Causes the names of the affected files to be printed. With the `-t` option, provides a detailed listing.

`-l` Whenever possible, link files rather than copy them. Usable only with the `-p` option.

`-m` Retain previous file modification time. This option is ineffective on directories that are being copied.

`-f` Copy in all files except those in *patterns.*

Example 1:

```
ls | cpio –oc >/rsave
```

Example 2:

```
cd $HOME/olddir
find .  –depth –print | cpio –pdl $HOME/newdir
```

SEE ALSO

ar(BU_CMD), find(BU_CMD), ls(BU_CMD), sh(BU_CMD), tar(AU_CMD).

LEVEL

Level 1.

NAME

ctags – create a tags file for use with ex and vi

SYNOPSIS

`ctags` [`–aBFtuwx`] [`–f` *tagsfile*] *filename* ...

DESCRIPTION

The `ctags` command makes a tags file for `ex` [see ex(AU_CMD)] from the specified C, Pascal, FORTRAN, `yacc` [see yacc(SD_CMD)], and `lex` [see lex(SD_CMD)], sources. A tags file gives the locations of specified objects (in this case functions and type definitions) in a group of files. Each entry in the tags is composed of three fields separated by white space, the object name, the the file in which it is defined, and an address specification. Function definitions are located using regular expression patterns, type definitions, using a line number.

`ex` and `vi` [see vi(AU_CMD)] use entries in the tags file to locate and display a definition.

Normally `ctags` places the tag descriptions in a file called `tags`; this may be overridden with the `–f` option. By default, the tags file is sorted in lexicographic (ASCII) order, and `ex` expects its entries to be so sorted.

Files with names ending in `.c` or `.h` are assumed to be C source files and are searched for C routine and macro definitions. Files with names ending in `.y` are assumed to be `yacc` source files. Files with names ending in `.l` are assumed to be `lex` files. Others are first examined to see if they contain any Pascal or FORTRAN routine definitions; if not, they are processed again looking for C definitions.

The tag for the `main()` function is treated specially in C programs. The tag formed is created by prepending `M` to *filename,* with a trailing `.c` removed, if any, and leading pathname components also removed. This makes use of `ctags` practical in directories with more than one program.

The options have the following meanings:

`–a` Append output to an existing tags file. The resulting file is not sorted. To preserve the order, use `–u` instead.

`–B` Use backward searching patterns (`?` ... `?`).

`–F` Use forward searching patterns (`/` ... `/`) (default).

`–t` Create tags for typedefs.

`–u` Update the specified files in the tags file. Entries that refer to them are deleted and then replaced in lexicographic order. Beware: this option is implemented in a way which is rather slow; it may be faster simply to rebuild the tags file.

`–w` Suppress warning diagnostics.

`–x` Produce a list of object names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

FILES

`tags`	output tags file

USAGE

End-user.

Recognition of functions, subroutines and procedures for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if there are two Pascal procedures in different blocks with the same name, `ctags` will only make an entry for one.

`ctags` does not know about `#ifdefs`.

`ctags` should know about Pascal types. It relies on the input being well formed to detect typedefs. Use of `-tx` shows only the last line of typedefs.

SEE ALSO

ex(AU_CMD), find(BU_CMD), vi(AU_CMD), lex(SD_CMD, yacc(SD_CMD)

LEVEL

Level 1.

NAME

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** *char***] [-s] [***file* **. . .]**

DESCRIPTION

Use **cut** to cut 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, (**-c** or **-b** options) or the length can vary from line to line and be marked with a field delimiter character like *tab* (**-f** option). **cut** can be used as a filter; if no files are given, the standard input is used. A file name of ''**-**'' explicitly refers to standard input.

cut processes supplementary code set characters, and recognizes supplementary code set characters in the *char* given to the **-d** option (see below) according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar**(BA_ENV)].

The *list* argument is a comma or **blank** separated list of positive integer field numbers or ranges. In either case the field numbers start at 1. Ranges can take one of the three forms below, where *number* is an unsigned integer.

*number***-***number*
Represents all the fields from the first *number* to the second *number*.

*number***-**
Represents all the fields from *number* to the last field, inclusive.

-*number*
Represents all the fields from the first to *number*, inclusive.

The elements in the list can be repeated, can overlap, and can be specified in any order. It is not an error to select fields that are not present in an input line. See the USAGE section for examples.

The meanings of the options are:

-b *list* The fields specified in *list* represent bytes. The selected bytes are written to the output, unless **-n** option is also specified.

-n Do not split characters. When the **-b** option is specified with this option, each element in *list* of the form *low***-**

When *list* is in the form *low***-**, each element in *list* is treated as described for *low***-** *high* above, with *high* set to the number of bytes in the current line, excluding the terminating **newline** character.

When *list* is in the form **-** *high*, each element in *list* is treated like *low***-***high* above, with *low* set to 1.

When *list* is given as a single number, *num*, each element is treated like *low***-** *high* above with *low* and *high* set to *num*.

-c *list* The fields specified in *list* represent characters. (This differs from **-b** because a single character may be many bytes.)

-f *list* The fields specified in *list* represent fields assumed to be separated in the file by a delimiter character (see **-d**). Lines with no field delimiters will be passed through intact (useful for table subheadings), unless **-s** is specified. Output fields are separated by a single occurrence of the delimiter character.

-d *char* The field delimiter used by **-f** is **char**. The default is *tab.* Space or other characters with special meaning to the shell must be quoted. *char* may be a supplementary code set character.

-s Suppresses lines with no delimiter characters when used with the **-f** option.

USAGE

Any of the following strings are legal *lists*: ''**1,4,7**'' (copies fields 1, 4 and 7 only); ''**1-3 8**'' (copies fields 1, 2, 3, and 8 only); ''**-5,10**'' (short for ''**1-5,10**''); or ''**3-**'' (short for third through last field). Note that **blank** separated lists need to be surrounded by quotes on the command line.

The following are some useful examples:

cut -d: -f1,5 /etc/passwd mapping of user login IDs to names

SEE ALSO

grep(BU_CMD), **paste**(BU_CMD)

LEVEL

Level 1.

NOTICES

Use **grep**(BU_CMD) 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**.

NAME

date – print and set the date

SYNOPSIS

date [-u] [+*format***]**

date [-u] [*mmddHHMM***[[***cc***]***yy***]]**

date [-a [-]*sss.fff***]**

DESCRIPTION

If no argument is given, or if the argument begins with **+**, the current date and time are printed. Otherwise, the current date is set if the user is a privilege user.

Supplementary code set characters in **+** *format* (see below) are recognized and displayed according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar**(BA_ENV)]. Month and weekday names are recognized according to the locale specified in the **LC_TIME** environment variable, as described below.

-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 Coordinated Universal Time or Greenwich Mean Time, bypassing the normal conversion to (or from) local time.

mm is the month number

dd is the day number in the month

HH is the hour number (24 hour system)

MM is the minute number

cc is the century minus one

yy is the last 2 digits of the year number

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

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. 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 a privileged 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 **LANG** on **envvar**(BA_ENV)].

+ *format* If the argument begins with **+**, the output of **date** is under the control of the user. Each Field Descriptor is preceded by **%** and is replaced in the output by its corresponding value. A single **%** is encoded by **%%**. All other characters are copied to the output

without change. The string is always terminated with a new-line character. If the argument contains embedded blanks it must be quoted (see the EXAMPLE section). Supplementary code set characters may be used in *format*.

As noted, month and weekday names are recognized according to the locale specified in the **LC_TIME** environment variable [see **LANG** on **envvar**(BA_ENV)]. The names are taken from a file whose format is specified in **strftime**(BA_LIB). This file also defines country-specific date and time formats such as **%c**, which specifies the default date format. The following form is the default for **%c**:

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

For example: Fri Dec 23 10:10:42 EST 1992

Field Descriptors (must be preceded by a **%**):

- **a** abbreviated weekday name
- **A** full weekday name
- **b** abbreviated month name
- **B** full month name
- **c** country-specific date and time format
- **C** century as a decimal integer (equivalent to the year divided by 100)
- **d** day of month – 01 to 31
- **D** date as **%m/%d/%y**
- **e** day of month – 1 to 31 (single digits are preceded by a blank)
- **h** abbreviated month name (alias for **%b**)
- **H** hour – 00 to 23
- **I** hour – 01 to 12
- **j** day of year – 001 to 366
- **m** month of year – 01 to 12
- **M** minute – 00 to 59
- **n** insert a new-line character
- **p** string containing ante-meridian or post-meridian indicator (by default, AM or PM)
- **r** time as **%I:%M:%S %p**
- **R** time as **%H:%M**
- **S** second – 00 to 61, allows for leap seconds
- **t** insert a tab character
- **T** time as **%H:%M:%S**
- **U** week number of year (Sunday as the first day of the week) – 00 to 53
- **w** day of week – Sunday = 0
- **W** week number of year (Monday as the first day of the week) – 00 to 53
- **x** country-specific date format
- **X** country-specific time format
- **y** year within century – 00 to 99
- **Y** year as *ccyy* (4 digits)
- **Z** abbreviated timezone name

Some of the field descriptors above can be modified by prepending them with an **E** or **O**, resulting in the following descriptors.

Modified Field Descriptors (must be preceded by a **%**):

- **Ec** Alternate appropriate date and time representation.
- **EC** Name of the base year (period) in locale's alternate representation.
- **Ex** Locale's alternate date representation.
- **Ey** Offset from %EC (year only) in locale's alternate representation.
- **EY** Full alternate year representation.
- **Od** Day of month using locale's alternate numeric symbols.
- **Oe** Day of month using locale's alternate numeric symbols.
- **OH** Hour (24 hr clock) using locale's alternate numeric symbols.
- **OI** Hour (12 hr clock) using locale's alternate numeric symbols.
- **Om** Month using locale's alternate numeric symbols.
- **OM** Minutes using locale's alternate numeric symbols.
- **OS** Seconds using locale's alternate numeric symbols.
- **OU** Week number of year (Sunday is 1st day of week) using locale's alternate numeric symbols.
- **Ow** Week day as number in locale's alternate representation (Sunday = 0).
- **OW** Week number of year (Monday is 1st day of week) using locale's alternate numeric symbols.
- **Oy** Year (offset from %C) in alternate representation.

USAGE

Examples

The command

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

generates as output:

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

SEE ALSO

envvar (BA_ENV), **strftime** (BA_LIB)

LEVEL

Level 1.

NAME

defadm – display / modify default values

SYNOPSIS

defadm
defadm [*filename* [*name* [=*value*]] [*name* [=*value*]] [**...**]]
defadm [**–d** *filename name* [*name*] [**...**]]

DESCRIPTION

The command **defadm** prints, modifies, adds, or deletes system default values contained in the file given by *filename*; where *filename* is a file in the **/etc/default** directory.

If **defadm** is executed with no arguments, it lists the files in the **/etc/default** directory.

If **defadm** is given a *filename*, it prints all the defaults contained in the *filename*. If one or more *names* are specified together with a *filename*, then **defadm** prints the specified *names* together with their *values*.

If **defadm** is given a *filename* and one or more *names* and *values*, then the default file is modified so that the *name* is set to the respective *value*.

If *defadm* is given the **-d** option, a *filename*, and one or more *names*, it removes the specified *names* from the *filename*.

FILES

/etc/default/audit,
/etc/default/cron,
/etc/default/login,
/etc/default/passwd,
/etc/default/su,
/etc/default/useradd,
/etc/default/userdel.

SEE ALSO

cron(AU_CMD), passwd(AU_CMD), su(AU_CMD), useradd(AS_CMD), userdel(AS_CMD).

LEVEL

Level 1.

NAME

df – report number of free disk blocks and i-nodes

SYNOPSIS

`df` [`-F` *FSType*] [`-begklntV`] [`-o` *specific_options*] [*directory*|*special* ...]

DESCRIPTION

`df` is a file system-independent command which prints the allocation portions of the generic superblock for mounted file systems, directories or mounted resources by examining the information returned by `statvfs()` [see statvfs(BA_OS)]. *directory* represents a valid directory name. `df` reports on the device that contains the directory. If *directory* represents a mount point then `df` reports on the file system mounted on the mount point. *special* represents a special device (*e.g.*, `/dev/dsk/c1d0s8`). *specific_options* represent options specified as a comma separated list of keywords and/or keyword-attribute pairs which are to be interpreted by the *FSType*-specific module.

The options have the following meaning:

`-F` 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`.

`-o` specify *FSType*-specific options, if any.

`-b` Print only the number of kilobytes free.

`-e` Print only the number of files free.

`-g` Print the entire `statvfs` structure. Used only for mounted file systems. Cannot be used with the `-o` option. This option will override the `-b`, `-e`, `-k`, `-n`, and `-t` options.

`-l` Report on local file systems only. Used only for mounted file systems. Cannot be used with the `-o` option.

`-n` Print only the *FSType* name. Invoked with no arguments, this option prints a list of mounted file system types. Used only for mounted file systems. Cannot be used with the `-o` option.

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

`-k` Print allocation in kilobytes. Because its output format is different from that of other options, this option should only be invoked with the `-l` or `-V` options.

`-V` Echo complete command line but do not execute command. The command line is generated by using the options and arguments provided, plus determining the others by a `/etc/mnttab` or `/etc/vfstab` lookup.

If no arguments or options are specified, the free space on all local and remotely mounted file systems is printed.

FILES

`/dev/dsk/*`

`/etc/mnttab`

`/etc/vfstab` table of file system information

USAGE

General

SEE ALSO

mount(AS_CMD).

FUTURE DIRECTIONS

Note that the current System V output format is preserved for compatibility reasons only and will be phased out in a future release.

LEVEL

Level 1.

NAME

diff – differential file comparator

SYNOPSIS

diff [-bitw] [-c | -e | -f | -h | -n] *filename1 filename2*

diff [-bitw] [-C *number***]** *filename1 filename2*

diff [-bitw] [-D *string***]** *filename1 filename2*

diff [-bitw] [-c | -e | -f | -h | -n] [-l] [-r] [-s] [-S *name***]** *directory1 directory2*

DESCRIPTION

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 dates are in the format that output from **date "+%a %b %e %T %Y"** produces. This is affected by the **LC_TIME** environment variable. [see **date**(BU_CMD) and **LANG** on **envvar**(BA_ENV)]. The lines removed from *filename1* are marked with '—'; those added to *filename2* 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. The form of *number* is affected by the **LC_NUMERIC** environment variable. [see **LANG** on **envvar**(BA_ENV)].

-e Produces a script of *a, c,* and *d* commands for the editor **ed**, which will recreate *filename2* from *filename1*. In connection with **-e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version **ed** scripts ($2,$3, . . .) made by **diff** need be on hand. A ''latest version'' appears on the standard output.

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

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 **-e** and **-f** are unavailable with **-h**.

-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 *filename1* and *filename2* with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *filename1*, while defining *string* will yield *filename2*.

The following options are used for comparing directories:

-l Produce output in long format. Before the **diff**, each text file is piped through **pr**(BU_CMD) 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 identical; these would not otherwise be mentioned.

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

Errors

The exit status returns **0** if no differences are found, **1** if differences are found, and **2** if an error occurred.

SEE ALSO

cmp (BU_CMD), **comm** (BU_CMD), **ed** (BU_CMD), **pr** (BU_CMD)

LEVEL

Level 1.

NOTICES

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

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [-exEX3] *file1 file2 file3*

DESCRIPTION

diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	*file1* is different
====2	*file2* is different
====3	*file3* is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

f **:** *n1* **a**	Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.
f **:** *n1* **,** *n2* **c**	Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n*2, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

-e Produce a script for the editor **ed**(BU_CMD) that will incorporate into *file1* all changes between *file2* and *file3*, that is, the changes that normally would be flagged **====** and **====3**.

-x Produce a script to incorporate only changes flagged **====**.

-3 Produce a script to incorporate only changes flagged **====3**.

-E Produce a script that will incorporate all changes between *file2* and *file3*, but treat overlapping changes (that is, changes that would be flagged with **====** in the normal listing) differently. The overlapping lines from both files will be inserted by the edit script, bracketed by **<<<<<<** and **>>>>>>** lines.

-X Produce a script that will incorporate only changes flagged **====**, but treat these changes in the manner of the **-E** option.

The following command will apply the resulting script to *file1*.

(cat script; echo '1,$p') | ed - *file1*

SEE ALSO

diff (BU_CMD)

LEVEL

Level 1

NAME

du – estimate file space usage

SYNOPSIS

`du [-ars] [`*file* ...]

DESCRIPTION

The command `du` gives an estimate, in 512-byte units, of the file space contained in all the specified files. Whenever a directory is named, all files within it are reported; sub-directories are traversed recursively. If no *file* is specified, the current directory is used.

The option `–s` causes only the grand total (for each of the specified *file*s) to be given. The option `–a` causes a report to be generated for each file. With no options, a report is given for each directory only.

`du` is normally silent about directories that cannot be read, files that cannot be opened, etc. The `–r` option will cause `du` to generate messages in such instances.

A file with two or more links is only counted once.

USAGE

General.

Sparse files may generate incorrect (high) estimates.

LEVEL

Level 1.

NAME

echo – echo arguments

SYNOPSIS

echo [*arg***] . . .**

echo [*arg***]**

DESCRIPTION

echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It processes supplementary code set characters according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar**(BA_ENV)].

The **/usr/bin/sh** version understands the following C-like escape conventions; beware of conflicts with the shell's use of ****:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
****	backslash
\0*n*	where *n* is the 1-, 2-, or 3-digit octal encoding of an 8-bit character. Each byte of multibyte characters should be preceded by backslash (\).
-n	do not add the newline to the output.

echo is useful for producing diagnostics in command files, for sending known data into a pipe, and for displaying the contents of environment variables.

SEE ALSO

sh (BU_CMD)

LEVEL

Level 1.

NAME

ed, **red** – text editor

SYNOPSIS

ed [-s] [-p *string***] [***file***]**

red [-s] [-p *string***] [***file***]**

DESCRIPTION

ed is the standard text editor. If the *file* argument is given, **ed** simulates an **e** command (see below) on the named file; that is to say, the file is read into **ed**'s buffer so that it can be edited. Both **ed** and **red** process supplementary code set characters in *file*, and recognize supplementary code set characters in the prompt string given to the **-p** option (see below) according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar**(BA_ENV)]. In regular expressions, pattern searches are performed on characters, not bytes, as described below.

-s Suppresses the printing of byte counts by **e**, **r**, and **w** commands, of diagnostics from **e** and **q** commands, and of the **!** prompt after a **!***shell command*.

-p Allows the user to specify a prompt string. The string may contain supplementary code set characters.

ed operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a **w** (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

red is a restricted version of **ed**. It will only allow editing of files in the current directory. It prohibits executing shell commands via **!***shell command*. Attempts to bypass these restrictions result in an error message (restricted shell).

Both **ed** and **red** support the **fspec** formatting capability. After including a format specification as the first line of *file* and invoking **ed** with your terminal in **stty –tabs** or **stty tab3** mode [see **stty**(AU_CMD)], [see **stty**(1)], the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

```
<:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: when you are entering text into the file, this format is not in effect; instead, because of being in **stty –tabs** or **stty tab3** mode, tabs are expanded to every eighth column.

Commands to **ed** have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While **ed** is accepting text, it is said to be in *input mode*. In this mode, no commands are recognized; all input is merely collected. Leave input mode by typing a period (**.**) at the beginning of a line, followed immediately by pressing RETURN.

ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (for example, **s**) to specify portions of a line that are to be substituted. 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. The regular expressions allowed by **ed** are constructed as follows:

The following one-character regular expressions match a single character:

1.1 An ordinary character (not one of those discussed in 1.2 below) is a one-character regular expression that matches itself.

1.2 A backslash (\) followed by any special character is a one-character regular expression 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 a regular expression (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 a regular expression (see 4.2 below).

d. The character that is special for that specific regular expression, that is used to bound (or delimit) a regular expression. (For example, see how slash (**/**) is used in the **g** command, below.)

1.3 A period (**.**) is a one-character regular expression that matches any character, including supplementary code set characters, except new-line.

1.4 A non-empty string of characters enclosed in square brackets (**[ ]**) is a one-character regular expression that matches one character, including supplementary code set characters, in that string. If, however, the first character of the string is a circumflex (^), the one-character regular expression matches any character, including supplementary code set characters, 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, including supplementary code set characters; for example, **[0-9]** is equivalent to **[0123456789]**. Characters specifying the range must be from the same code set; when the characters are from different code sets, one of the characters specifying the range is matched. 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 regular expressions from one-character regular expressions:

2.1 A one-character regular expression is a regular expression that matches whatever the one-character regular expression matches.

2.2 A one-character regular expression followed by an asterisk (*****) is a regular expression that matches zero or more occurrences of the one-character regular expression, which may be a supplementary code set character. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character regular expression followed by **\{***m***\}**, **\{***m*,**\}**, or **\{***m*,*n***\}** is a regular expression that matches a range of occurrences of the one-character regular expression. 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 regular expression matches as many occurrences as possible.

2.4 The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.

2.5 A regular expression enclosed between the character sequences **\(** and **\)** is a regular expression that matches whatever the unadorned regular expression 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 regular expression. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of **\(** counting from the left. For example, the expression **^\(.*\)\1$** matches a line consisting of two repeated appearances of the same string.

A regular expression may be constrained to match words.

3.1 **\<** constrains a regular expression 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 regular expression must be a digit, underscore, or letter.

3.2 **\>** constrains a regular expression to match the end of a string or to precede a character that is not a digit, underscore, or letter.

A regular expression 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 a regular expression constrains that regular expression to match an initial segment of a line.

4.2 A dollar sign (**$**) at the end of an entire regular expression constrains that regular expression to match a final segment of a line.

4.3 The construction **^***regular expression***$** constrains the regular expression to match the entire line.

The null regular expression (for example, **//**) is equivalent to the last regular expression encountered. See also the last paragraph of the DESCRIPTION section below.

To understand addressing in **ed** it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character **.** addresses the current line.
2. The character **$** addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. *'x* addresses the line marked with the mark name character *x*, which must be a lower-case letter (**a**–**z**). Lines are marked with the **k** command described below.
5. A regular expression 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 regular expression. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph of the DESCRIPTION section below.
6. A regular expression 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 regular expression. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph of the DESCRIPTION section below.
7. An address followed by a plus sign (**+**) or a minus sign (**-**) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. 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 **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (**,**). They may also be separated by a semicolon (**;**). In the latter case, the first address is calculated, the current line (.) is set to that value, and then the second address is calculated. This feature can be used to determine the starting line for forward and backward searches (see Rules 5 and 6, above). The second address of any two-address sequence must correspond to a line in the buffer that follows the line corresponding to the first address.

In the following list of **ed** commands, the parentheses shown prior to the command are not part of the address; rather they show the default address(es) for the command.

The *file* arguments of the **e**, **E**, **f**, **r**, **w**, and **W** commands are subject to pattern matching as in **sh**(BU_CMD), They should be separated from the command letter by one or more spaces or tabs.

It is generally illegal for more than one command to appear on a line. However, any command (except **e**, **f**, **r**, or **w**) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively, as discussed below under the **l**, **n**, and **p** commands.

(.)a
<text>
.
The **a**ppend command accepts zero or more lines of text and appends it after the addressed line in the buffer. The current line (**.**) is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the ''appended'' text to be placed at the beginning of the buffer. The maximum number of bytes that may be entered from a terminal is 256 per line. {LINE_MAX} per line (including the newline character). {LINE_MAX} is defined in **limits.h**.

(.)c
<text>
.
The **c**hange command deletes the addressed lines from the buffer, then accepts zero or more lines of text that replaces these lines in the buffer. The current line (**.**) is left at the last line input, or, if there were none, at the first line that was not deleted.

(. , .)d
The **d**elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*
The **e**dit command deletes the entire contents of the buffer and then reads the contents of *file* into the buffer. The current line (**.**) is set to the last line of the buffer. If *file* is not given, the currently remembered file name, if any, is used (see the **f** command). The number of characters read in is printed; *file* is remembered for possible use as a default file name in subsequent **e**, **r**, and **w** commands. If *file* is replaced by **!**, the rest of the line is taken to be a shell [**sh**(1)] command whose output is to be read in. Such a shell command is not remembered as the current file name. See also DIAGNOSTICS below.

E *file* The **E**dit command is like **e**, except that the editor does not check to see if any changes have been made to the buffer since the last **w** command.

f *file* If *file* is given, the **f**ile-name command changes the currently remembered file name to *file*; otherwise, it prints the currently remembered file name.

(1,$)g/*regular expression***/***command list*
In the **g**lobal command, the first step is to mark every line that matches the given regular expression. Then, for every such line, the given *command list* is executed with the current line (**.**) initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; **a**, **i**, and **c** commands and associated input are permitted. The **.** terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the **p** command. The **g**, **G**, **v**, and **V** commands are not permitted in the *command list*. See the NOTICES section and the last paragraph of the DESCRIPTION section below.

(1,$)G/*regular expression***/**
In the interactive **G**lobal command, the first step is to mark every line that matches the given regular expression. Then, for every such line, that line is printed, the current line (**.**) is changed to that line, and any one command (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an **&** causes the re-execution of the most recent command executed within the current invocation of **G**. Note that the commands input as part of the execution of the **G** command may address and affect any lines in the buffer. The **G** command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h The **h**elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H The **H**elp command causes **ed** to enter a mode in which error messages are printed for all subsequent **?** diagnostics. It will also explain the previous **?** if there was one. The **H** command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.
The **i**nsert command accepts zero or more lines of text and inserts it before the addressed line in the buffer. The current line (**.**) is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the **a** command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line. {LINE_MAX} per line (including the new-line character). {LINE_MAX} is defined in **limits.h**.

(.,.+1)j
The **j**oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*
The **mar**k command marks the addressed line with name *x*, which must be a lower-case letter (**a-z**). The address *'x* then addresses this line; the current line (**.**) is unchanged.

(.,.)l
The **l**ist command prints the addressed lines in an unambiguous way: a few non-printing characters (for example, tab, backspace) are represented by visually mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. All other non-printing characters are printed in octal, long lines are folded and the end of each line is marked with a **$**. An **l** command may be appended to any command other than **e**, **f**, **r**, or **w**.

(.,.)m*a*
The **m**ove command repositions the addressed line(s) after the line addressed by *a*. Address **0** is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; the current line (**.**) is left at the last line moved.

(.,.)n
The **n**umber command prints the addressed lines, preceding each line by its line number and a tab character; the current line (**.**) is left at the last line printed. The **n** command may be appended to any command other than **e**, **f**, **r**, or **w**.

(.,.)p
The **p**rint command prints the addressed lines; the current line (**.**) is left at the last line printed. The **p** command may be appended to any command other than **e**, **f**, **r**, or **w**. For example, **dp** deletes the current line and prints the new current line.

P The editor will prompt with a ***** for all subsequent commands. The **P** command alternately turns this mode on and off; it is initially off.

q The **q**uit command causes **ed** to exit. No automatic write of a file is done; however, see DIAGNOSTICS below.

Q The editor exits without checking if changes have been made in the buffer since the last **w** command.

($)r *file*
The **r**ead command reads the contents of *file* into the buffer. If *file* is not given, the currently remembered file name, if any, is used (see the **e** and **f** commands). The currently remembered file name is not changed unless *file* is the very first file name mentioned since **ed** was invoked. Address 0 is legal for **r** and causes the file to be read in at the beginning of the buffer. If the read is successful, the number of characters read in is printed; the current line (**.**) is set to the last line read in. If *file* is replaced by **!**, the rest of the line is taken to be a shell [see **sh**(BU_CMD)] command whose output

is to be read in.

For example, **$r !ls** appends current directory to the end of the file being edited. Such a shell command is not remembered as the current file name.

(.,.)s/*regular expression***/***replacement***/** or
(.,.)s/*regular expression***/***replacement***/g** or
(.,.)s/*regular expression***/***replacement***/***n* *n* = 1–512

The **s**ubstitute command searches each addressed line for an occurrence of the specified regular expression. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number *n*, appears after the command, only the *n*-th occurrence of the matched string on each addressed line is replaced. It is an error if the substitution fails on all addressed lines. Any character other than space or new-line may be used instead of **/** to delimit the regular expression and the *replacement*; the current line (**.**) is left at the last line on which a substitution occurred. See also the last paragraph of the DESCRIPTION section below.

An ampersand (**&**) appearing in the *replacement* is replaced by the string matching the regular expression on the current line. The special meaning of **&** in this context may be suppressed by preceding it by ****. As a more general feature, the characters *****n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified regular expression enclosed between **\(** and **\)**. When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of **\(** starting from the left. When the character **%** is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The **%** loses its special meaning when it is in a replacement string of more than one character or is preceded by a ****.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by ****. Such substitution cannot be done as part of a **g** or **v** command list.

(.,.)t*a*

This command acts just like the **m** command, except that a copy of the addressed lines is placed after address **a** (which may be 0); the current line (**.**) is left at the last line copied.

u The **u**ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **v**, **G**, or **V** command.

(1,$)v/*regular expression***/***command list*

This command is the same as the global command **g**, except that the lines marked during the first step are those that do not match the regular expression.

(1,$)V/*regular expression***/**
This command is the same as the interactive global command **G**, except that the lines that are marked during the first step are those that do not match the regular expression.

(1,$)w *file*
The **w**rite command writes the addressed lines into *file*. If *file* does not exist, it is created with mode **666** (readable and writable by everyone), unless your file creation mask dictates otherwise; see the description of the **umask** special command on **sh**(BU_CMD). The currently remembered file name is not changed unless *file* is the very first file name mentioned since **ed** was invoked. If no file name is given, the currently remembered file name, if any, is used (see the **e** and **f** commands); the current line (**.**) is unchanged. If the command is successful, the number of characters written is printed. If *file* is replaced by **!**, the rest of the line is taken to be a shell command whose standard input is the addressed lines. Such a shell command is not remembered as the current file name.

(1,$)W *file*
This command is the same as the **w**rite command above, except that it appends the addressed lines to the end of *file* if it exists. If *file* does not exist, it is created as described above for the **w** command.

($)= The line number of the addressed line is typed; the current line (**.**) is unchanged by this command.

!*shell command*
The remainder of the line after the **!** is sent to the UNIX system shell to be interpreted as a command. See **sh**(BU_CMD). Within the text of that command, the unescaped character **%** is replaced with the remembered file name; if a **!** appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, **!!** will repeat the last shell command. If any expansion is performed, the expanded line is echoed; the current line (**.**) is unchanged.

(.+1)<new-line>
An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, **ed** prints a **?** and returns to its command level.

Some size limitations: 512 bytes in a line, 256 bytes in a global command list and in the pathname of a file (counting slashes). {LINE_MAX} bytes in a line, 256 bytes in a global command list, and {PATH_MAX} bytes in the pathname of a file (counting slashes). {LINE_MAX} and {PATH_MAX} are defined in **limits.h**. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, **ed** discards ASCII NUL characters.

If a file is not terminated by a new-line character, **ed** adds one and puts out a message explaining what it did.

If the closing delimiter of a regular expression or of a replacement string (for example, **/**) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

```
s/s1/s2        s/s1/s2/p
g/s1           g/s1/p
?s1            ?s1?
```

Errors

? for command errors.

?*file* for an inaccessible file.
(use the **h**elp and **H**elp commands for detailed explanations).

If changes have been made in the buffer since the last **w** command that wrote the entire buffer, **ed** warns the user if an attempt is made to destroy **ed**'s buffer via the **e** or **q** commands. It prints **?** and allows one to continue editing. A second **e** or **q** command at this point will take effect. The **-s** command-line option inhibits this feature.

SEE ALSO

ex(AU_CMD), **grep**(BU_CMD), **sed**(BU_CMD), **sh**(BU_CMD), **stty**(AU_CMD) **umask**(BU_CMD), **vi**(AU_CMD)

LEVEL

Level 1.

NOTICES

The **-** option, although it continues to be supported, has been replaced in the documentation by the **-s** option that follows the Command Syntax Standard A **!** command cannot be subject to a **g** or a **v** command.

The **!** command and the **!** escape from the **e**, **r**, and **w** commands cannot be used if the editor is invoked from a restricted shell [see **sh**(BU_CMD)].

The sequence **\n** in a regular expression does not match a new-line character.

If the editor input is coming from a command file (for example, **ed** *file* **<** *ed_cmd_file*), the editor exits at the first failure.

The following environment variables affect the execution of **pr**: **LANG**, **LC_ALL**, **LC_CTYPE**, **LC_MESSAGES**, **LC_TIME**, **TZ** [see **LANG** on **envvar**(BA_ENV)].

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr *arguments*

DESCRIPTION

The *arguments* are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2s complement numbers. The length of the expression is limited to 512 characters. Expressions may be grouped using (escaped) parentheses.

The operators and keywords are listed below. Characters that need to be escaped in the shell [see **sh** (BU_CMD)] are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within **{ }** symbols.

expr **\|** *expr*
Return the first *expr* if it is neither null nor **0**, otherwise return the second *expr*.

expr **\&** *expr*
Return the first *expr* if neither *expr* is null or **0**, otherwise return **0**.

expr **{ =, \>, \>=, \<, \<=, != }** *expr*
Return the result of an integer comparison if both arguments are integers, otherwise return the result of a lexical comparison.

expr **{ +, - }** *expr*
Add or subtract integer-valued arguments.

expr **{ *, /, % }** *expr*
Multiply, divide, or compute remainder of integer-valued arguments.

expr **:** *expr*
match *expr expr*
Compare the first argument with the second argument, which must be a regular expression. Regular expression syntax is the same as that of **ed** (BU_CMD) except that all patterns are ''anchored'' (that is, begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the **\(. . . \)** pattern symbols can be used to return a portion of the first argument.

length *string*
Return the length of *string*.

substr *string index count*
Return the portion of *string* composed of at most *count* characters starting at the character position of *string* as expressed by *index* (where the first character of *string* is index 1, not 0).

index *string character_sequence*
Return the index of the first character in *string* that is also in *character_sequence* or 0 to indicate no match.

expr processes supplementary code set characters according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar** (BA_ENV). In regular expressions, pattern searches are performed on characters, not bytes, as described on **ed** (BU_CMD)

Errors

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

0 The expression is neither null nor **0**.
1 The expression is null or **0**.
2 An expression is invalid.

non-numeric argument arithmetic attempted on a non-numeric string

USAGE

Examples

Add 1 to the shell variable **a**:

```
a=‘expr $a + 1‘
```

The following example emulates **basename** (BU_CMD) 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**. The **//** characters eliminate any ambiguity about the division operator.

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

SEE ALSO

ed(BU_CMD), **sh**(BU_CMD)

NOTICES

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

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

looks like:

```
expr = = =
```

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

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

LEVEL

Level 1.

NAME

file – determine file type

SYNOPSIS

`file` [`-f` *file*] [`-h`] *file* ...

DESCRIPTION

The command `file` performs a series of tests on each specified *file* in an attempt to classify it. If it appears to be a text file, `file` examines an initial segment and makes a guess about its language. (The answer is not guaranteed to be correct.) If *file* is an executable ("`a.out`") file it is identified as such, and any other available information is reported.

If the `-f` option is given, the next argument is taken to be a file containing the names of the files to be examined.

If the argument is a symbolic link, by default, the link is followed and `file` tests the file that the symbolic link references. When the `-h` is specified, symbolic links will not be followed and the output will be:

`symbolic link to` *pathname*

where *pathname* is the pathname of the referenced file.

LEVEL

Level 1.

NAME

find – find files

SYNOPSIS

`find` *path-name-list expression*

DESCRIPTION

The command `find` recursively descends the directory hierarchy for each pathname in the *path-name-list* (that is, one or more pathnames) seeking files that match a boolean *expression* written in the primaries given below. In the following descriptions, the argument *n* is used as a decimal integer where `+`*n* means more than *n*, `-`*n* means less than *n*, and *n* means exactly *n*.

The *expression* argument is made up of:

`-name` *pattern*
: True if *pattern* matches the current filename. Normal shell filename generation characters [see sh(BU_CMD)] 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.

`-perm 0`*num*
: True if the file permission flags exactly match the octal number 0*num* [see chmod(BU_CMD)]. If 0*num* is prefixed by a minus sign, only the bits that are set in 0*num* are compared with the file permission flags, and the expression evaluates true if they match.

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

`-follow`
: Always true; causes symbolic links to be followed. When following symbolic links, `find` keeps track of the visited directories so it can detect any infinite loops that would be caused by a symbolic link pointing up to an ancestor. This expression should not be used with the `-type l` option.

`-links` *n*
: True if the file has *n* links.

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

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

`-level [``=|-|+`**`]`***level*
: True if the file level exactly matches (`'='`), dominates (`'+'`), or is dominated by (`-`) level. The level can be specified as a fully qualified level, alias name, or a level identifier (LID) [see lvlname(ES_CMD)].

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

`-atime` *n*
: True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by `find` itself.

`-mtime` *n*
: True if the file has been modified in *n* days.

`-ctime` *n*
: True if the file inode has been changed in *n* days.

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

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

`-print` Always true; causes the current pathname to be printed.

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

`-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` [see cpio(BU_CMD)] to transfer files that are contained in directories without write permission.

`-fstype` *type*
: True if the file system to which the file belongs is of type *type*.

`-inum` *n* True if the file has inode number *n*.

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

`-nogroup`
: True if the file belongs to a group not in the `/etc/group` file.

`-prune` Always yields true. Has the side effect of pruning the search tree at the file.

expression True if the parenthesized expression is true (parentheses must be escaped if they are special to the command interpreter).

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

1. The negation of a primary (`!` is the unary *not* operator).
2. Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

3. Alternation of primaries (`-o` is the OR operator).

FILES

`/etc/passwd`
`/etc/group`

USAGE

General.

When `find` is used in conjunction with `cpio` and the command to follow symbolic links is applied to one of them (`-L` for `cpio` and `-follow` for `find`), then the other must also follow symbolic links. Applying a 'follow symbolic links' option to one command and not the other causes unexpected and unusual results.

EXAMPLE

To remove all files named `tmp` or ending in `.xx` that have not been accessed for a week:

```
find / \(-name tmp -o -name '*.xx' \) -atime+7 -exec rm {} \;
```

SEE ALSO

chmod(BU_CMD), cpio(BU_CMD), sh(BU_CMD), stat(BA_OS), test(BU_CMD).

LEVEL

Level 1.

NAME

fmt – simple text formatters

SYNOPSIS

fmt [-cs] [-w *width***] [***file* **. . .]**

DESCRIPTION

fmt is a simple text formatter that fills and joins lines to produce output lines of (up to) the number of characters specified in the **-w** *width* option. The default *width* is 72. **fmt** concatenates the *inputfile*s listed as arguments. If none are given, **fmt** formats text from the standard input.

Blank lines are preserved in the output, as is the spacing between words. **fmt** does not fill lines beginning with a ''**.**'' (dot), for compatibility with **nroff**. Nor does it fill lines starting with ''**From:**''.

Indentation is preserved in the output, and input lines with differing indentation are not joined (unless **-c** is used).

fmt can also be used as an in-line text filter for **vi**(AU_CMD); the **vi** command:

```
!}fmt
```

reformats the text between the cursor location and the end of the paragraph.

OPTIONS

-c Crown margin mode. Preserve the indentation of the first two lines within a paragraph, and align the left margin of each subsequent line with that of the second line. This is useful for tagged paragraphs.

-s Split lines only. Do not join short lines to form longer ones. This prevents sample lines of code, and other such formatted text, from being unduly combined.

-w *width* Fill output lines to up to *width* columns.

SEE ALSO

vi (AU_CMD)

LEVEL

Level 1.

NOTICES

The **-w** *width* option is acceptable for BSD compatibility, but it may go away in future releases.

NAME

fmtmsg – display a message in the standard format on standard error and the system console

SYNOPSIS

`fmtmsg` [`-c` *classification*] [`-u` *subclass*] [`-l` *label*] [`-s` *severity*] [`-t` *tag*] [`-a` *action*] *text*

DESCRIPTION

Based on a message's classification component, `fmtmsg` either writes a formatted message to standard error, to the console, or to both.

A formatted message consists of up to five standard components as defined below. The components, *classification* and *subclass*, are not part of the standard message displayed to the user, but define the source of the message and direct the display of the formatted message.

Options

`-c` *classification*
: Describes the source of the message. Valid keywords are:

  `hard` indicates that the source of the condition is hardware
  `soft` indicates that the source of the condition is software
  `firm` indicates that the source of the condition is firmware

`-u` *subclass*
: A comma list of keywords that further defines the message and directs the display of the message. Valid keywords are:

  `appl` identifies the condition as having originated in an application. This keyword should not be used in combination with either `util` or `opsys`.
  `util` identifies the condition as having originated in a utility. This keyword should not be used in combination with either `appl` or `opsys`.
  `opsys` identifies the message as having originated in the kernel. This keyword should not be used in combination with either `appl` or `util`.
  `recov` indicates that the application will recover from the condition. This keyword should not be used in combination with `nrecov`.
  `nrecov` indicates that the application will not recover from the condition. This keyword should not be used in combination with `recov`.
  `print` causes the message to be printed to the standard error stream. `print`, `console`, or both may be used.
  `console` causes the message to be written to the system console.

`-l` *label*
: Identifies the source of the message.

`-s` *severity*
: Indicates the seriousness of the error. The keywords and definitions of the standard levels of *severity* are:

`halt`	indicates that the application has encountered a severe fault and is halting.
`error`	indicates that the application has detected a fault.
`warn`	indicates a condition that is out of the ordinary and might be a problem.
`info`	provides information about a condition that is not in error.

`-t` *tag* — The string containing an identifier for the message.

`-a` *action* — A text string describing the first step in the error-recovery process. This string must be written so that the entire *action* argument is interpreted as a single argument. `fmtmsg` precedes each action string with the prefix: `TO FIX:`.

text — A text string describing the condition. Must be written so that the entire *text* argument is interpreted as a single argument.

ERRORS

The exit codes for `fmtmsg` are the following:

`0` = All the requested functions were executed successfully.
`1` = The command contains a syntax error, an invalid option, or an invalid argument to an option.
`2` = The command executed with partial success, however the message was not displayed on standard error.
`4` = The command executed with partial success, however the message was not displayed on the system console.
`32` = No requested functions were executed successfully.

USAGE

There are two environment variables that control the behavior of `fmtmsg`: `MSGVERB` and `SEV_LEVEL`. `MSGVERB` is set by the administrator in the `/etc/profile` for the system. Users can override the system-set `MSGVERB` by resetting `MSGVERB` in their own `.profile` files or by changing the value in their current shell session. `SEV_LEVEL` can be used in shell scripts.

`MSGVERB` tells `fmtmsg` which message components to select when writing messages to standard error. The value of `MSGVERB` is a colon list of optional keywords. `MSGVERB` can be set as follows:

```
MSGVERB=[keyword[:keyword]...]
export MSGVERB
```

Valid *keywords* are: `label`, `severity`, `text`, `action` and `tag`. If `MSGVERB` contains a keyword for a component and the component's value is not the null value, `fmtmsg` includes that component in the message when writing the message to standard error. If `MSGVERB` does not include a keyword for a message component, that component is not included in the display of the message. The keywords may appear in any order. If `MSGVERB` is not defined, if its value is the null string, if its value is not of the correct format, or if it contains keywords other than the valid ones listed above, `fmtmsg` selects all components.

MSGVERB affects only which message components are selected for display. All message components are included in console messages.

SEV_LEVEL defines severity levels and associates print strings with them for use by fmtmsg. The standard severity levels shown below cannot be modified. Additional severity levels can be defined, redefined, and removed.

keyword	*level*	*print string*
(none)	0	(no severity is used)
halt	1	HALT
error	2	ERROR
warn	3	WARNING
info	4	INFO

SEV_LEVEL is set as follows:

SEV_LEVEL=[*description*[**:***description*]...]
export SEV_LEVEL

The format of *description* is a three-field comma list as follows:

description=severity_keyword,level,print_string

severity_keyword
: a character string that is used as the keyword on the –s *severity* option to fmtmsg.

level
: a character string that evaluates to a positive integer (other than 0, 1, 2, 3, or 4, that are reserved for the standard severity levels). If the keyword *severity_keyword* is used, *level* is the severity value passed on to fmtmsg().

print_string
: the character string used by fmtmsg in the standard message format whenever the severity value *level* is used.

If SEV_LEVEL is not defined, or if its value is null, no severity levels other than the defaults are available. If a *description* in the colon list is not a three-field comma list, or if the second field of a comma list does not evaluate to a positive integer, that *description* in the colon list is ignored.

EXAMPLE

Example 1:

The following example of fmtmsg produces a complete message in the standard message format and displays it to the standard error stream:

```
$ fmtmsg –c soft –u recov,print,appl –l UX:cat \
> –s error -t UX:cat:138 \
> –a "refer to manual" "invalid syntax"
```

produces:

```
UX:cat: ERROR: invalid syntax
       TO FIX: refer to manual  UX:cat:138
```

Example 2:

When the environment variable MSGVERB is set as follows:

```
MSGVERB=severity:text:action
```

and Example 1 is used, fmtmsg produces:

```
 ERROR: invalid syntax
TO FIX: refer to manual
```

Example 3:

When the environment variable SEV_LEVEL is set as follows:

```
SEV_LEVEL=note,5,NOTE
```

the following fmtmsg command:

```
$ fmtmsg –c soft –u print –l UX:cat –s note \
> –a "refer to manual" "invalid syntax"
```

produces:

```
UX:cat: NOTE: invalid syntax
      TO FIX: refer to manual
```

and displays the message on standard error.

SEE ALSO

fmtmsg(BA_LIB).

FUTURE DIRECTIONS

This command is to be removed when the three year waiting period has expired.

LEVEL

Level 2: April 1991.

NAME

gettxt – retrieve a text string from a message data base

SYNOPSIS

`gettxt` *msgfile*`:`*msgnum* [*dflt_msg*]

DESCRIPTION

`gettxt` retrieves a text string from a message file in the directory `/usr/lib/locale/`*locale*`/LC_MESSAGES`. The directory *locale* corresponds to the language in which the text strings are written [see setlocale(BA_OS)].

msgfile Name of the file in the directory:
`/usr/lib/locale/`*locale*`/LC_MESSAGES`
from which *msgnum* is to be retrieved. The name of *msgfile* can be up to 14 characters in length, but may not contain either \0 (null) or the ASCII codes for `/` (slash) or `:` (colon).

msgnum Sequence number of the string to retrieve from *msgfile*. The strings in *msgfile* are numbered sequentially from *1* to *n*, where *n* is the number of strings in the file.

dflt_msg Default string to be used on failure to retrieve the message from the file.

The text string to be retrieved is in the file created by the `mkmsgs` utility and installed in the *locale* directory in `/usr/lib/locale/`*locale*`/LC_MESSAGES`. The environment variable `LC_MESSAGES` controls which directory is searched. If `LC_MESSAGES` is not set, the environment variable `LANG` will be used. If `LANG` is not set, the language in which the strings will be retrieved is U. S. English and the files containing the strings are in `/usr/lib/locale/C/LC_MESSAGES/*` .

If `gettxt` fails to retrieve a message in requested language, it will try to retrieve the same message in the C locale. If this also fails, the processing depends on the second argument. If the second argument is not supplied on the command line or it is the null string, `gettxt` will display the string `Message not found!!` The second argument will be displayed if it is not the null string.

Nongraphic characters may be included in the default message as alphabetic escape sequences.

EXAMPLE

```
gettxt UX:10 "hello world\n"
```

FILES

`/usr/lib/locale/C/LC_MESSAGES/*` default message files created by `mkmsgs`

`/usr/lib/locale/`*locale*`/LC_MESSAGES/*` message files for different languages created by `mkmsgs`

SEE ALSO

gettxt(BA_LIB), mkmsgs(AS_CMD), setlocale(BA_OS), srchtxt(AS_CMD).

LEVEL

Level 1.

NAME

grep – search a file for a pattern

SYNOPSIS

grep [-E | -F] [-c | -l | -q] [-bhinsvx] *expression* **[***file* **...]**

grep [-E | -F] [-c | -l | -q] [-bhinsvx] -e *expression* **...**
[-f *exprfile***] ... [***file* **...]**

grep [-E | -F] [-c | -l | -q] [-bhinsvx] [-e *expression***] ...**
-f *exprfile* **... [***file* **...]**

DESCRIPTION

grep searches files for patterns and prints all lines that contain at least one of the patterns. By default **grep** uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with **ed**(BU_CMD) to match the patterns (in *expression* and *exprfile*). It uses a compact non-deterministic algorithm. If the **-E** or **-F** options are specified, **grep** behaves like **egrep** or **fgrep**, see **OPTIONS**.

Be careful using the characters **$**, ∗, **[**, ^, |, **(**, **)**, and \ in the *expression* because they are also meaningful to the shell. It is safest to enclose the entire *expression* in single quotes ′ . . . ′. A null *expression* matches all lines.

grep processes supplementary code set characters according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar**(BA_ENV)]. except as noted under the **-i** option below. The pattern searches are performed on characters, not bytes, as described in **ed**(BU_CMD).

If no files are specified, **grep** assumes standard input. If a ‘‘**-**’’ is specified as a file, standard input is used. Normally, each line matched is copied to standard output. The filename is printed before each line matched if there is more than one input file.

OPTIONS

-E Specifying **-E** on a **grep** command line, is equivalent to calling **egrep**. All specified patterns (in *expression* and *exprfile*) are then full regular expressions. See **egrep**(AU_CMD) for an explanation of full regular expressions. When this option is specified, all other **grep** options (except **-F**) have the same effect as usual, and the same effect as they have for **egrep**(AU_CMD).

-F Specifying **-F** on a **grep** command line, is equivalent to calling **fgrep**. All specified patterns (in *expression* and *exprfile*) are then character strings. When this option is specified, all other **grep** options (except **-E**) have the same effect as usual, and the same effect as they have for **fgrep**.

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

-e *expression*
Specify one or more patterns (regular expressions or strings) to be used during the search for input. The patterns in *expression* are separated by **newline** characters. Two adjacent **newline**s indicate a null pattern. The last pattern does not require a terminating **newline**. When multiple **-e** or **-f** options are specified, all the patterns specified will be used. (Obviously, if *expression* is to contain **newline**s, it should be quoted.)

This option is useful for specifying patterns that begin with a ''**-**''.

-f *exprfile*
Read one or more patterns (regular expressions or strings) from *exprfile*. The patterns in *exprfile* are terminated by a **newline** character. An empty line in *exprfile* indicates a null pattern. When multiple **-e** or **-f** options are specified, all the patterns specified will be used.

-h Suppress printing of filenames when searching multiple files.

-i Ignore uppercase/lowercase distinction during comparisons, as defined by the locale in **LC_CTYPE** [see **LANG** on **envvar**(BA_ENV)]. Valid for single-byte characters only.

-l Print the names of files with matching lines once, separated by newlines. Does not repeat the names of files when a pattern is found more than once. If the input file is **stdin**, then a message such as ''**(standard input)**'' will be written, depending upon the message locale.

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

-q Quiet, do not write anything to the standard output, regardless of any matches. Exits with zero if any input line is matched.

-s Suppress error messages about nonexistent or unreadable files.

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

-x Match only lines for which the pattern matches the entire line. For character strings, the pattern must match all characters in the line. For regular expressions, this option is equivalent to placing a ''**^**'' at the start of the pattern, and a ''**$**'' at the end of the pattern.

Errors

Exit status returns **0** if any matches are found, **1** if none are found, and **2** for syntax errors or inaccessible files (even if matches were found).

SEE ALSO

ed (BU_CMD), **egrep** (AU_CMD), **sed** (BU_CMD), **sh** (BU_CMD)

LEVEL

Level 1.

NOTICES

Lines are limited to **LINE_MAX** bytes; longer lines are truncated. **LINE_MAX** is defined in **/usr/include/limits.h**.

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.

NAME

head – display first few lines of files

SYNOPSIS

head [-n *number***] [***file* **. . .]**

head [-*number***] [***file* **. . .]**

DESCRIPTION

head copies the first *number* lines of each *file* to the standard output. If no *file* is given, **head** copies lines from the standard input. The environment variable **LC_CTYPE** is referenced so that the codesets used in the data are processed correctly. [see **LANG** on **envvar**(BA_ENV)]. The default value of *number* is 10 lines.

When more than one file is specified, the start of each file will look like:
==>*file***<==**

Thus, a common way to display a set of short files, identifying each one, is:

head -n 9999 *file1 file2* **...**

SEE ALSO

cat(BU_CMD), **more**(BU_CMD), **pg**(BU_CMD), **tail**(BU_CMD)

LEVEL

Level 1.

NOTICES

The **-***number* option has been made obsolete by POSIX, hence it is recommended that application authors avoid its use in favor of the **-n** *number* option.

The length of the input lines is limited to {**LINE_MAX**} bytes.

NAME

kill – send a signal to a process

SYNOPSIS

kill [-s *signal***]** *pid* **. . .**

kill -l [*status***]**

kill [-*signal***]** *pid* **. . .**

kill -l

DESCRIPTION

kill sends a signal to the specified processes. The value of *signal* may be numeric or symbolic [see **signal**(BA_OS)]. The symbolic signal name is the name as it appears in **/usr/include/sys/signal.h**, with the **SIG** prefix stripped off. Signal 15 (**SIGTERM**) is sent by default; this will normally kill processes that do not catch or ignore the signal.

pid is either an unsigned or negative integer that identifies which process(es) should receive the signal. If *pid* is unsigned, the process with process ID *pid* is selected. If *pid* is preceded by a negative sign (**-**), all processes with process group ID *pid* are selected.

For example, if *pid* is **0**, all processes in the process group are signaled.

The signaled process must belong to the current user unless the user is a privileged user.

The process number of each asynchronous process started with **&** is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using **ps**(BU_CMD).

Options

-s *signal* Send *signal* to the selected processes.

-l [*status***]** If *status* is null, print a list of symbolic signal names that may be used as *signal*. If *status* is not null, it is either a return status from a process terminated by a signal (stored in the **?** environment variable for the most recently completed process), or a signal number. In both cases the symbolic name of the matching signal is printed.

-*signal* Send *signal* to the selected processes. This option is the same as **-s** *signal*.

-l Print a list of symbolic signal names that may be used as *signal*.

SEE ALSO

kill(BA_OS), **ps**(BU_CMD), **sh**(BU_CMD), **signal**(BA_OS), **signal**(BA_ENV)

LEVEL

Level 1.

NOTICES

The **-***signal* usage is for backward compatibility, and may not be supported in future releases. It should therefore be avoided.

NAME

lfmt – display error message in standard format and pass to logging and monitoring services

SYNOPSIS

`lfmt [-c][-f` *flags*`][-l` *label*`][-s` *severity*`][-g` *catalog*`:`*msgnum*`]` *format* `[`*args*`]`

DESCRIPTION

lfmt uses *format* for **printf** style formatting of *args*. **lfmt** encapsulates the output in the standard error message format and displays the output on *stderr*. In addition, **lfmt** forwards its output to the logging and monitoring facility.

The following options are available.

-c Display the output on the console, with a date and time stamp.

-f *flags* Specify logging information as a comma-separated list of keywords from the following sets:

Major classification
Identifies the source of the condition. Identifiers are: **hard** (hardware), **soft** (software), and **firm** (firmware).

Message source subclassification
Identifies the type of software in which the problem is spotted. Identifiers are: **appl** (application), **util** (utility), and **opsys** (operating system).

-g *catalog*`:`*msgid*
Specify that a localized version of *format* should be retrieved from a locale-specific message database. *catalog* indicates the message database that contains the localized version of the *format* string. *catalog* is limited to 14 characters. These characters must be selected from a set of all character values, excluding **\0** (null) and the ASCII codes for **/** (slash) and **:** (colon).

msgid is a positive number that indicates the index of the string into the message database.

If *catalog* does not exist in the current locale (identified by the **LC_MESSAGES** or **LANG** environment variables), or if *msgid* is out of bounds, **lfmt** will attempt to retrieve the message from the C locale. If this second retrieval fails, **lfmt** uses the *format* string as passed on the command line.

lfmt will output **Message not found!!\n** as the *format* string if *catalog* is not a valid catalog name as defined above and *msgid* is not a positive number.

-l *label* Specify the label string to be displayed with the message (e.g., **"UX:cat"**). *label* is a character string no more than 25 characters in length; it will be automatically suffixed with a colon (**:**). When unspecified, no label is displayed as part of the message.

-s *severity*

Specify the severity string to be displayed with the message. Acceptable strings include the standard severities in either their print string (i.e., **HALT**, **ERROR**, **INFO**, **WARNING**, and **"TO FIX"**) or keyword (i.e., **halt**, **error**, **info**, **warn**, and **action**) forms, or any other user-defined string. The severity will be suffixed with a colon (**:**). The **ERROR** severity will be used if no severity is specified.

Standard Error Message Format

lfmt displays error messages in the following format:

*label***:** *severity***:** *text*

If no *label* was defined using the **-l** *label* option, the message is displayed in the format:

*severity***:** *text*

If **lfmt** is called twice to display an error message and a helpful *action* or recovery message, the output can look like the following:

*label***:** *severity***:** *text*
*label***: TO FIX:** *text*

ERRORS

Upon success, lfmt exits with code 0. Upon failure, lfmt exits with the following codes:

1 write error.

2 cannot log or forward to console.

3 syntax error.

EXAMPLE

Example 1:

```
lfmt -f soft,util -l UX:test -s info "test facility enabled"
```

displays the message to *stderr* and makes it available for logging:

```
UX:test: INFO: test facility enabled
```

SEE ALSO

envvar(BA_ENV), gettxt(BU_CMD), pfmt(BU_CMD), pfmt(BA_LIB), lfmt(BA_LIB), printf(BU_CMD).

LEVEL

Level 2, April 1991.

NAME

line – read one line

SYNOPSIS

line

DESCRIPTION

line copies one line (up to a newline) from the standard input and writes it on the standard output. It returns an exit code of 1 on **EOF** and always prints at least a newline. It is often used within shell files to read from the user's terminal.

The **LC_CTYPE** environment variable defines the processing of the codesets used in the input file. [See **LANG** in **envvar**(BA_ENV).]

SEE ALSO

read (BA_OS) **sh** (BU_CMD)

LEVEL

Level 1.

NAME

listusers – list user information

SYNOPSIS

```
listusers [-h] [-v]
listusers [-g groups] [-l logins]
```

DESCRIPTION

Executed without any options, this command lists all user logins sorted by login. The output shows the login ID and the account field value in `/etc/passwd`.

`-h` Prints the valid security levels of the invoking user. This option is only valid if the enhanced security extension is implemented.

`-v` Prints the default security level of the invoking user. This option is only valid if the enhanced security extension is implemented.

`-g` Lists all users belonging to *group*, sorted by login. Multiple groups can be specified as a comma-separated list.

`-l` Lists the user or users specified by *logins*, sorted by login. Multiple logins can be specified as a comma-separated list.

USAGE

The `-l` and `-g` options can be combined. Users will only be listed once, even if they belong to more than one of the selected groups.

LEVEL

Level 1.

NAME

ln – link files

SYNOPSIS

ln -s [-f] [-n] *file1* **[***file2* **...]** *target*

DESCRIPTION

The **ln** command links *filen* to *target* by creating a directory entry that refers to *target*. By using **ln** with one or more file names, the user may create one or more links to *target*.

The **ln** command may be used to create both hard links and symbolic links; by default it creates hard links. A hard link to a file is indistinguishable from the original directory entry. Any changes to a file are effective independent of the name used to reference the file. Hard links may not span file systems and may not refer to directories.

Without the **-s** option, **ln** is used to create hard links. *filen* is linked to *target*. If *target* is a directory, another file named *filen* is created in *target* and linked to the original *filen*. If *target* is a file that already exists, **ln** will print an error and go onto the next *filen* (unless **-f** is specified).

There are three options to **ln**. If multiple options are specified, the one with the highest priority is used and the remainder are ignored. The options, in descending order of priority, are:

-s **ln** will create a symbolic link. A symbolic link contains the name of the file to which it is linked. Symbolic links may span file systems and may refer to directories. If the linkname exists, then do not overwrite the contents of the file. A symbolic link's permissions are always set to read, write, and execute permission for owner, group, and world (**777**).

-f **ln** will link files without generating any errors, even if the mode of the file *target* forbids writing. Note, however, that if *target* refers to a directory that has no write permissions, errors will still occur.

-n If the linkname is an existing file, do not overwrite the contents of the file. The **-f** option overrides this option.

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. *filen* 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 *filen* or the last component of *filen*. This file is a symbolic link that references *filen*. If *target* does not exist, a file with the name *target* is created and it is a symbolic link that references *filen*.

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 *filen*, a file is created in *target* whose name is *filen* or its last component; each new *filen* is a symbolic link to the original *filen*. The *files* and *target* may reside on different file systems.

SEE ALSO

chmod (BU_CMD), **cp** (BU_CMD), **link** (BA_OS), **rm** (BU_CMD), **stat** (BA_OS), **symlink** (BA_OS)

FUTURE DIRECTIONS

This page is Level 2 to allow for the future **-n** option.

LEVEL

Level 2.

NOTICES

Doing operations that involve ''**..**'' (such as ''**cd ..**'') in a directory that is symbolically linked will reference the original directory not the target.

The **-s** option does not use the current working directory. In the command

ln -s *path target*

path is taken literally without being evaluated against the current working directory.

If the POSIX2 environment variable is set and exported, the behavior of **ln** with no options is the same as the current **ln -n**.

The **-n** option is for backward compatibility only. It should not be used, since it may be removed in future releases.

Use the **POSIX2** environmental variable to get POSIX.2 behavior that is inconsistent with existing System V behavior. **POSIX2** requires no prompting in case of existing target.

NAME

ls, **lc** – list contents of directory

SYNOPSIS

ls [-abCcdeFfgiLlmnopqRrstux1] [*file* **. . .]**

lc [-1CFLabcfgilmnopqRrstux] [*name* **. . .]**

DESCRIPTION

For each directory argument, **ls** lists the contents of the directory; for each *file* argument, **ls** repeats its name and any other information requested. The output is sorted alphabetically by default. When arguments are not 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. **ls** processes supplementary code set characters according to the locale specified in the **LC_CTYPE** and **LC_COLLATE** environment variables [see **LANG** on **envvar**(BA_ENV)]. except as noted under the **-b** and **-q** options below.

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 options **-C** and **-x** enable multi-column formats; and the **-m** option enables stream output format, in which files are listed across the page, separated by commas.

To determine output formats for the **-C**, **-x**, and **-m** options, **ls** uses an environment variable, **COLUMNS**, to determine the number of positions available on one output line. If this variable is not set, the **terminfo** 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 **ls** command has the following options:

-a List all entries, including those that begin with a period (**.**), which are normally not listed.

-b Force printing of non-printable characters to be in the octal *****ddd* notation. All multibyte characters are considered printable.

-C Multi-column output with entries sorted down the columns. This is the default output format.

-c Use time of last modification of the i-node (file created, mode changed, and so on) for sorting (**-t**) or printing (**-l**).

-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.

-e *extent_opt*
Specify how to handle a file that has extent attribute information. Extent attributes include reserved space, a fixed extent size, and extent alignment. It may not be possible to preserve the information if the destination file system does not support extent attributes, has a different block size than the source file system, or lacks free extents appropriate to satisfy the extent attribute requirements. Valid values for *extent_opt* are:

warn Issue a warning message if extent attribute information cannot be kept (default).

force Fail the copy if extent attribute information cannot be kept.

ignore Ignore extent attribute information entirely.

When used with **-l**, **-e** displays extent attribute information for files with reserved space or fixed extent sizes.

-F Put a slash (**/**) after each filename if the file is a directory, an asterisk (*) if the file is executable, a vertical bar (**|**) if it is a FIFO, and an at sign (**@**) if the file is a symbolic link.

-f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **-l**, **-t**, **-s**, and **-r**, and turns on **-a**; the order is the order in which entries appear in the directory.

-g 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 When listing status, if an argument is a symbolic link, list the status of the file or directory referenced by the link rather than that of the link itself.

-l List in long format, giving mode, number of links, owner, group, size in bytes, and date and time of last modification for each file. The date is given in the locale specified by the **LC_TIME** environmental variable. If the file is a special file, the size field contains the major and minor device numbers rather than a size. If the file is a symbolic link, the filename is printed followed by ''**->**'' and the pathname of the referenced file.

-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 (**?**). All multibyte characters are considered printable.

-R Recursively list subdirectories encountered.

-r Reverse the order of sort to get the reverse of the locale's collation sequence or oldest first as appropriate.

-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 **-n** 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.

-z Print the alias of the security level for each file. (Valid only if the Enhanced Security Utilities are installed.)

-Z Print the fully qualified security level for each file. (Valid only if the Enhanced Security Utilities are installed.)

The **-z** and **-Z** options are mutually exclusive. If the **-z** option is specified and there is not an alias assigned to the level, the decimal value of the level identifier (LID) is displayed. If the **-z** or **-Z** option is specified and the level is in the *valid-inactive* state, the decimal value of the LID is displayed. [See **lvlname**(1M) for a description of LID states.]

The mode printed under the **-l** option consists of eleven possible characters. The first character may be one of the following:

d if the entry is a directory;
l if the entry is a symbolic link;
b if the entry is a block special file;
c if the entry is a character special file;
p if the entry is a fifo (named pipe) special file;
- if the entry is a regular 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, write, and 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.

ls -l (the long list) prints its output as follows:

```
-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 bytes. The owner of the file, or the user, belongs to the group **dev** (perhaps indicating ''development''), and their login name is **smith**. The number, in this case **1**, indicates the number of links to file **part2** [see **cp**(BU_CMD)]. 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

- **l** mandatory locking occurs during access (the set-group-ID bit is on and the group execution bit is off)
- **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**(BU_CMD)], and execution is on
- **T** the 1000 bit is turned on, and execution is off (undefined bit-state)

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 other permissions, the third position may be occupied by **t** or **T**. These refer to the state of the sticky bit and execution permissions.

The **-e** option (used with **-l**) displays extent attribute information as follows:

```
-rwxrwxrwx 1 smith dev  10876  May 16 9:42 part2 :res 36 ext 3 align noextend
```

This output line indicates a file with 36 blocks of reservation, a fixed extent size of 3 blocks, all extents aligned to 3 block boundaries, and the file unable to be grown once the current reservation is exhausted.

The format of the date given in the long listing is dependent on the environment variable **LC_TIME** (the examples are in the **C** locale). [see **LANG** on **envvar**(BA_ENV)].

USAGE

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:

```
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:

```
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.

Files

/etc/passwd	user IDs for **ls -l** and **ls -o**
/etc/group	group IDs for **ls -l** and **ls -g**
/usr/share/lib/terminfo/?/*	terminal information database

SEE ALSO

chmod (BU_CMD), **find** (BU_CMD), **getacl** (ES_CMD), **lvlname** (ES_CMD), **setacl** (ES_CMD)

LEVEL

Level 1.

NAME

mail, rmail – send or read mail

SYNOPSIS

`mail` [`-epqr`] [`-f`*file*]

`mail` [`-t`] *name* ...

`rmail` [`-t`] *name* ...

DESCRIPTION

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

<newline>
: Go on to next message.

`+` Same as <newline>.

`d` Delete message and go on to next message.

`p` Print message again.

`–` Go back to previous message.

`s` [*file*]
: Save message in the named *file* (`mbox` is default).

`w` [*file*]
: Save message, without its header, in the named *file* (`mbox` is default).

`m` [*name* ...]
: Mail the message to the named users (names are user login names; the default is the user).

`q` Put undeleted mail back in the mailfile and stop.

`EOF` (Usually CTRL-D.) Same as `q`.

`x` Put all mail back in the mailfile unchanged and stop.

`!`*command*
: Escape to the command interpreter to execute *command*.

`*` Print a command summary.

The optional arguments alter the printing of the mail:

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

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

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

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

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

When *name*s (user login names) are given, `mail` takes the standard input up to an end-of-file (or up to a line consisting of just a `.`) and adds it to each recipient's mailfile. The message is preceded by three lines:

- a line containing the sender's name and a postmark,
- a line of the form "Content-Length: XXXXX", and
- a blank line,

where the XXXXX on the Content-Length: line represents the number of characters in the message after the blank line. The `-t` option causes the message to be preceded by all users the mail is sent to. If a user being sent mail is not recognized, or if `mail` is interrupted during input, the file `dead.letter` will be saved to allow editing and resending. Note that this is regarded as a temporary file in that it is recreated every time needed, erasing the previous contents of `dead.letter`.

The command `rmail` only permits the sending of mail.

FILES

`/etc/passwd`	to identify sender and locate persons
`$HOME/mbox`	saved mail
`dead.letter`	unmailable text

SEE ALSO

sh(BU_CMD).

LEVEL

Level 1.

NAME

make – maintain, update, and regenerate groups of programs

SYNOPSIS

make [-f *makefile***] [-eiknpPqrstuw] [***names***]**

DESCRIPTION

make allows the programmer to maintain, update, and regenerate groups of computer programs. **make** executes commands in *makefile* to update one or more target *names* (*names* are typically programs). If the **-f** option is not present, then **makefile**, **Makefile**, and the Source Code Control System (SCCS) files **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is **-**, the standard input is taken. More than one **-f** *makefile* argument pair may appear.

make updates a target only if its dependents are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be outdated.

The following list of directives can be included in makefiles to modify the behavior of **make**. They are used in makefiles as if they were targets:

.DEFAULT: If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.

.IGNORE: Same effect as the **-i** option.

.MUTEX: Serialize the updating of specified targets (see the ''Parallel make'' subsection, below).

.PRECIOUS:
Dependents of the **.PRECIOUS** entry will not be removed when quit or interrupt are pressed.

.SILENT: Same effect as the **-s** option.

The options for **make** are listed below:

-e Environment variables override assignments within makefiles.

-f *makefile* Description filename (*makefile* is assumed to be the name of a description file).

-i Ignore error codes returned by invoked commands.

-k Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.

-n No execute mode. Print commands, but do not execute them. Even command lines beginning with an **@** are printed.

-p Print out the complete set of macro definitions and target descriptions.

-P Update in parallel more than one target at a time. The number of targets updated concurrently is determined by the environment variable **PARALLEL** and the presence of **.MUTEX** directives in makefiles.

-q Question. **make** returns a zero or non-zero status code depending on whether or not the target file has been updated.

-r Do not use the built-in rules.

-s Silent mode. Do not print command lines before executing.

-t Touch the target files (causing them to be updated) rather than issue the usual commands.

-u Unconditionally **make** the target, ignoring all timestamps.

-w Suppress warning messages. Fatal messages will not be affected.

Creating the makefile

The makefile invoked with the **-f** option (or accessed by default) is a carefully structured file of explicit instructions for updating and regenerating programs, and contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a **:**, then a (possibly null) list of prerequisite files or dependencies. Text following a **;** and all following lines that begin with a tab are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or **#** begins a new dependency or macro definition. Shell commands may be continued across lines with a backslash-new-line (\ new-line) sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (#) and new-line surround comments including contained \ new-line sequences.

The following makefile says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

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

Command lines are executed one at a time, each by its own shell. The **SHELL** environment variable can be used to specify which shell **make** should use to execute commands. The default is **/usr/bin/sh**. The first one or two characters in a command can be the following: **@**, **-**, **@-**, or **-@**. If **@** is present, printing of the command is suppressed. If **-** is present, **make** ignores an error. A line is printed when it is executed unless the **-s** option is present, or the entry **.SILENT:** is included in the makefile, or unless the initial character sequence contains a **@**. The **-n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see the discussion of the **MAKEFLAGS**

macro in the Environment section below). The **-t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate **make**. If the **-i** option is present, if the entry **.IGNORE:** is included in the makefile, or if the initial character sequence of the command contains **-**, the error is ignored. If the **-k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the directive **.PRECIOUS**.

Parallel make

If **make** is invoked with the **-P** option, it tries to build more than one target at a time, in parallel. (This is done by using the standard UNIX system process mechanism which enables multiple processes to run simultaneously.) For the makefile shown in the example in the previous section, it would create processes to build **a.o** and **b.o** in parallel. After these processes were complete, it would build **pgm**.

The number of targets **make** will try to build in parallel is determined by the value of the environment variable **PARALLEL**. If **-P** is invoked, but **PARALLEL** is not set, then **make** will try to build no more than two targets in parallel.

You can use the **.MUTEX** directive to serialize the updating of some specified targets. This is useful when two or more targets modify a common output file, such as when inserting modules into an archive or when creating an intermediate file with the same name, as is done by **lex** and **yacc**. If the makefile in the previous section contained a **.MUTEX** directive of the form

```
.MUTEX: a.o b.o
```

it would prevent **make** from building **a.o** and **b.o** in parallel.

Environment

The environment is read by **make**. All variables are assumed to be macro definitions and are processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **-e** option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built-in rules.

The **MAKEFLAGS** environment variable is processed by **make** as containing any legal input option (except **-f**, **-p** and **-r**) defined for the command line. Further, upon invocation, **make** invents the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This feature proves very useful for ''super-makes''. In fact, as noted above, when the **-n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make -n** recursively on a whole software system to see what would have been executed. This result is possible because the **-n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This usage is one way of debugging all of the makefiles for a software project without actually doing anything.

Include Files

If the string **include** appears as the first seven letters of a line in a makefile, and is followed by a blank or a tab, the rest of the line is assumed to be a filename and will be read by the current invocation, after substituting for any macros.

Macros

Entries of the form *string1* **=** *string2* are macro definitions. *string2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of **$(***string1***[:***subst1***=[***subst2***]])** are replaced by *string2*. The parentheses are optional if a single-character macro name is used and there is no substitute sequence. The optional :*subst1=subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown in the Libraries section below.

Internal Macros

There are five internally maintained macros that are useful for writing rules for building targets.

$*** The macro **$* stands for the filename part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

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

$<** The **$< macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module that is outdated with respect to the target (the manufactured dependent file name). Thus, in the **.c.o** rule, the **$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

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

or:

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

$?** The **$? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are outdated with respect to the target, and essentially those modules that must be rebuilt.

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

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros, the meaning is changed to *directory part* for **D** and *file part* for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **$?**.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, and so on. If no update commands for such a file appear in the makefile, and if an inferable prerequisite exists, that prerequisite is compiled to make the

target. In this case, **make** has inference rules that allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c      .c~      .f      .f~      .s      .s~      .sh      .sh~      .C      .C~
.c.a    .c.o     .c~.a   .c~.c    .c~.o   .f.a     .f.o     .f~.a     .f~.f   .f~.o
.h~.h   .l.c     .l.o    .l~.c    .l~.l   .l~.o    .s.a     .s.o      .s~.a   .s~.o
.s~.s   .sh~.sh  .y.c    .y.o     .y~.c   .y~.o    .y~.y    .C.a      .C.o    .C~.a
.C~.C   .C~.o    .L.C    .L.o     .L~.C   .L~.L    .L~.o    .Y.C      .Y.o    .Y~.C
.Y~.o   .Y~.Y
```

The internal rules for **make** are contained in the source file **rules.c** for the **make** program. These rules can be locally modified. To print out the rules compiled into the **make** on any machine in a form suitable for recompilation, the following command is used:

```
make -pf - 2>/dev/null </dev/null
```

A tilde in the above rules refers to an SCCS file [see **sccsfile**(4)]. Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with the **make** suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (for example, **.c:**) is the definition of how to build *x* from *x***.c**. In effect, the other suffix is null. This feature is useful for building targets from only one source file, for example, shell procedures and simple C programs.

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant: the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

```
.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f .f~ .C
.C~ .Y .Y~ .L .L~
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

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

This abbreviation is possible because **make** has a set of internal rules for building files. The user may add rules to this list by simply putting them in the makefile.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to **cc**(1), **lex**(1), and **yacc**(1), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

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

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date
.c.a:
        $(CC) -c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $(<F:.c=.o)
        rm -f $(<F:.c=.o)
```

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

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

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object filenames (inside **lib**) whose C source files are outdated. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this transformation may become possible in the future.) Also note the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

FILES

[Mm]akefile
s.[Mm]akefile
/usr/bin/sh

/usr/lib/locale/*locale***/LC_MESSAGES/uxepu**
language-specific message file

SEE ALSO

cc(SD_CMD), **cd**(BU_CMD), **lex**(SD_CMD), **printf**(BA_LIB), **sh**(BU_CMD), **yacc**(SD_CMD)

LEVEL

Level 1.

NOTICES

Some commands return non-zero status inappropriately; use **-i** or the **-** command line prefix to overcome the difficulty.

Filenames with the characters **= : @** will not work. Commands that are directly executed by the shell, notably **cd**(1), are ineffectual across new-lines in **make**. The syntax **lib(file1.o file2.o file3.o)** is illegal. You cannot build **lib(file.o)** from **file.o**.

NAME

mkdir – make a directory

SYNOPSIS

`mkdir` [`-m` *mode*] [`-p`] [`-M`] [`-l` *level*] *dirname* `...`

DESCRIPTION

The command `mkdir` creates the specified directories. Standard entries, `.`, for the directory itself, and `..`, for its parent, are made automatically. `mkdir` cannot create these entries by name. Creation of a directory requires write permission in the parent directory. If the parent directory is a Multilevel Directory (MLD) and the process is in real mode [see `mldmode`(ES_CMD)], the directory created will be an effective directory. Note that unlike regular directories, effective directories do not inherit the default ACLs of the parent.

The owner ID and group ID of the new directories are set to the process's real user ID and group ID, respectively.

The `mkdir` command has the following options:

`-m` Specifies the *mode* to be used for new directories. Choices for *mode* can be found in `chmod`(BU_CMD).

`-p` Creates all non-existing parent directories first.

`-M` Specifies that *dirname* is a MLD. This option is only valid when the caller has appropriate privileges and the Enhanced Security Extension is implemented.

`-l` *level* Specifies that *level* is to be applied to the named directory. It must be either a valid level alias name or be in the following format:

h_name [`:`*c_name* [`,`*c_name*]]`...`]

where *h_name* and *c_name* are described in `lvlname`(ES_CMD). This level information is verified against the Level Translation Data Base (LTDB) and then placed on the named directory. This option is only valid when the Enhanced Security Extension is implemented.

RETURN VALUE

The command `mkdir` returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

EXAMPLE

To create the subdirectory structure `ltr/jd/jan`, type:

```
mkdir -p ltr/jd/jan
```

SEE ALSO

intro(BA_OS), mkdir(BA_OS), rm(BU_CMD), sh(BU_CMD), umask(BU_CMD).

LEVEL

Level 1.

NAME

more, page – browse or page through a text file

SYNOPSIS

`more` [`-cdflsu`] [*-lines*] [*+linenumber*] [*+/pattern*] [*filename* ...]

`page` [`-cdflsu`] [*-lines*] [*+linenumber*] [*+/pattern*] [*filename* ...]

DESCRIPTION

`more` 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, and prints `--More--` at the bottom of the screen. `more` provides a two-line overlap between screens for continuity. If `more` is reading from a file rather than a pipe, the percentage of characters displayed so far is also shown.

`more` 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.

`page` clears the screen before displaying the next screenful of text; it only provides a one-line overlap between screens.

`more` 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.

If the standard output is not a terminal, `more` acts just like `cat` except that a header is printed before each file in a series.

If no files are given on the command line, `more` reads from standard input.

`more` accepts the following options and arguments:

`-c` Clear before displaying. Using `-c` redraws the screen, instead of scrolling it, 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 option is helpful for inexperienced users.

`-f` Do not fold long lines. This option is useful when lines contain nonprinting characters or escape sequences.

`-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.

`-s` Squeeze. Replace multiple blank lines with a single blank line.

`-u` Suppress generation of underlining escape sequences. Normally, `more` handles underlining 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.

-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.

`more` uses the terminal's `terminfo` entry to determine its display characteristics, and looks in the environment variable `MORE` for any preset options [see terminfo(TI_ENV)]. For instance, to page through files using the `-c` mode by default, set the value of `MORE` to `-c`.

Command Descriptions

The commands take effect immediately; it is not necessary to type a carriage return. 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*%) ' message.

In the following commands, *i* is a numerical argument (`1` by default).

i<space> Display another screenful, or *i* more lines if *i* is specified.

i<return> Display another line, or *i* more lines, if specified.

i^D (CTRL-D) Display (scroll down) a number of lines equal to half the screen size. If *i* is given, the scroll size is set to *i* lines.

*i*d Same as ^D.

*i*z Same as <space>, except that *i*, if present, becomes the new default number of lines per screenful.

*i*s Skip *i* lines and then print a screenful.

*i*f Skip *i* screenfuls and then print a screenful.

i^B (CTRL-B) Skip back *i* screenfuls and then print a screenful.

b Same as ^B (CTRL-B).

q
Q Exit from `more`.

= Display the current line number.

v Drop into the `vi` editor [see vi(AU_CMD)] at the current line of the current file.

h Help. Give a description of all the `more` commands.

i/*pattern* Search for the *i*th occurrence of the regular expression *pattern*. Display the screenful starting two lines prior to the line that contains the *i*th 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 such 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.

*i*n Search for the *i*th occurrence of the last *pattern* entered.

´ 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.

!*command*
Invoke a shell (determined by the SHELL environment variable with sh the default) 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.

i:n Skip to the *i*th 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 *i*th 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.

:f Display the current filename and line number.

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

. Dot. Repeat the previous command.

^\ Halt a partial display of text. more stops sending output, and displays the usual --More-- prompt. Some output, however, is lost as a result.

FILES

/etc/terminfo	terminal data base
/usr/lib/more.help	help file

SEE ALSO

cat(BU_CMD), sh(BU_CMD), terminfo(TI_ENV), vi(AU_CMD).

LEVEL

Level 1.

NAME

nawk – pattern-directed scanning and processing language

SYNOPSIS

`nawk` [`-F` *fs*] [*prog*] [*inputfile* ...]

`nawk` [`-F` *fs*] [`-f` *progfile*] [*inputfile* ...]

DESCRIPTION

`nawk` scans each input *inputfile* for lines that match any of a set of patterns specified literally in *prog* or in a file specified as `-f` *progfile*. With each pattern there can be an associated action that will be performed when a line of a *inputfile* matches the pattern. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern. The inputfile name `-` means the standard input. Any *inputfile* of the form *var=value* is treated as an assignment, not a filename.

An input line is made up of fields separated by white space, or by the regular expression assigned to special variable `FS`. The `-F` *fs* option defines the input field separator as the regular expression *fs*.

The fields are denoted `$1`, `$2`, ...; `$0` refers to the entire line.

A pattern-action statement has the form

```
pattern { action }
```

A missing `{ action }` means print the line; a missing pattern always matches. Pattern-action statements are separated by newlines or semicolons.

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

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

Statements are terminated by semicolons, newlines or right braces. An empty *expression-list* stands for `$0`. String constants are quoted `" "`, with the usual C language escapes recognized within. Expressions take on string or numeric values as appropriate, and are built using the operators `+ - * / % ^` (exponentiation), and concatenation (indicated by a blank). The operators `! ++ -- += -= *= /= %= ^= > >= < <= == != ?:` are also available in expressions. Variables may be scalars, array elements (denoted *x*`[`*i*`]`) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for

a form of associative memory. Multiple subscripts such as [i,j,k] are permitted; the constituents are concatenated, separated by the value of SUBSEP.

The print statement prints its arguments on the standard output (or on a file if >*file* or >>*file* is present or on a pipe if |*cmd* is present), separated by the current output field separator, and terminated by the output record separator. *file* and *cmd* may be literal names or parenthesized expressions; identical string values in different statements denote the same open file. The printf statement formats its expression list according to the format [see printf(BA_LIB)]. The built-in function close(*expr*) closes the file or pipe *expr*.

The customary functions exp, log, sqrt, sin, cos, and atan2 are built in. Other built-in functions are:

length
: the length in characters of its argument taken as a string, or of $0 if no argument.

rand
: random number on (0,1)

srand
: sets the seed for rand

int
: truncates to an integer value

substr(*s*, *m*, *n*)
: the *n*-character substring of *s* that begins at position *m* counted from 1.

index(*s*, *t*)
: the position in *s* where the string *t* occurs, or 0 if it does not.

match(*s*, *r*)
: the position in *s* where the regular expression *r* occurs, or 0 if it does not. The variables RSTART and RLENGTH are set to the position and length of the matched string.

split(*s*, *a*, *fs*)
: splits 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.

sub(*r*, *t*, *s*)
: substitutes *t* for the first occurrence of the regular expression *r* in the string *s*. If *s* is not given, $0 is used. sub returns the number of replacements.

gsub
: same as sub except that all occurrences of the regular expression are replaced; gsub returns the number of replacements.

sprintf(*fmt*, *expr*, ...)
: the string resulting from formatting *expr* ... according to the printf format *fmt*

system(*cmd*)
: executes *cmd* and returns its exit status

The ''function'' `getline` sets `$0` to the next input record from the current input file; `getline` <*file* sets `$0` to the next record from *file*. `getline` *x* sets variable *x* instead. Finally, `cmd|getline` pipes the output of *cmd* into `getline`; each call of `getline` returns the next line of output from *cmd*. In all cases, `getline` returns `1` for a successful input, `0` for end of file, and `-1` for an error.

Patterns are arbitrary Boolean combinations (with `! || && ()`) of regular expressions and relational expressions. Regular expressions are as in `egrep` [see egrep(AU_CMD)]. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions, using the operators `~` and `!~`. */re/* is a constant regular expression; any string (constant or variable) may be used as a regular expression, except in the position of an isolated regular expression in a pattern.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines from an occurrence of the first pattern though an occurrence of the second.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression
expression in array-name
(expr,expr,...) in array-name

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

The special patterns `BEGIN` and `END` may be used to capture control before the first input line is read and after the last. `BEGIN` and `END` do not combine with other patterns.

Variable names with special meanings:

`FS`	regular expression used to separate fields; also settable by option `-F` *fs*.
`NF`	number of fields in the current record.
`NR`	ordinal number of the current record.
`FNR`	ordinal number of the current record in the current file.
`FILENAME`	the name of the current input file.
`RS`	input record separator (default newline).
`OFS`	output field separator (default blank).
`ORS`	output record separator (default newline).
`OFMT`	output format for numbers (default **%.6g**).
`SUBSEP`	separates multiple subscripts (default 034).
`ARGC`	argument count, assignable.

ARGV argument array, assignable; non-null members are taken as filenames.

Functions may be defined (at the position of a pattern-action statement) thus:

```
function foo(a, b, c) { ...; return x }
```

Parameters are passed by value if scalar and by reference if array name; functions may be called recursively. Parameters are local to the function; all other variables are global.

EXAMPLE

```
length > 72
```
Print lines longer than 72 characters.

```
{ print $2, $1 }
```
Print first two fields in opposite order.

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```
Same, with input fields separated by comma and/or blanks and tabs.

```
     { s += $1 }
END  { print "sum is", s, " average is", s/NR }
```
Add up first column, print sum and average.

```
/start/, /stop/
```
Print all lines between start/stop pairs.

```
BEGIN     {    # Simulate echo(1)
     for (i = 1; i < ARGC; i++) printf "%s ", ARGV[i]
     printf "\n"
     exit }
```

SEE ALSO

egrep(AU_CMD), lex(SD_CMD), printf(BA_LIB), sed(BU_CMD).

FUTURE DIRECTIONS

In the SVID, 4th Edition, awk has the functionality described above. nawk is now designated Level 2 in the SVID Fourth Edition and will be removed when the Level 2 period has expired.

LEVEL

Level 2, June 1993.

NAME

nl – line numbering filter

SYNOPSIS

`nl` [`-h`*type*] [`-b`*type*] [`-f`*type*] [`-v`*start#*] [`-i`*incr*] [`-p`]
[`-l`*num*] [`-s`*sep*] [`-w`*width*] [`-n`*format*] [`-d`*delim*] [*file*]

DESCRIPTION

The command `nl` reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

`nl` views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line	*Start of*
`\:\:\:`	header
`\:\:`	body
`\:`	footer

Unless otherwise specified, `nl` assumes the text being read is in a single logical page body.

Options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

`-b`*type* Specifies which logical page body lines are to be numbered. Recognized *type*s and their meaning are: `a`, number all lines; `t`, number lines with printable text only, where all characters from supplementary code sets are considered printable; `n`, no line numbering; `p`*exp*, number only lines that contain the regular expression specified in `exp` [see ed(BU_CMD)]. Default *type* for logical page body is `t` (text lines numbered).

`-h`*type* Same as `-b`*type* except for header. Default *type* for logical page header is `n` (no lines numbered).

`-f`*type* Same as `-b`*type* except for footer. Default for logical page footer is `n` (no lines numbered).

`-p` Do not restart numbering at logical page delimiters.

`-v`*start#* The initial value used to number logical page lines. Default is `1`.

`-i`*incr* The increment value used to number logical page lines. Default is `1`.

`-s`*sep* The character(s) used in separating the line number and the corresponding text line. These characters must be single-byte characters. Default *sep* is a tab.

`-w`*width* The number of columns to be used for the line number. Default *width* is `6`.

`-n`*format* The line numbering format. Recognized values are: `ln`, left justified, leading zeroes suppressed; `rn`, right justified, leading zeroes suppressed; `rz`, right justified, leading zeroes kept. Default *format* is `rn` (right justified).

`-l`*num* The number of blank lines to be considered as one. For example, `-l2` results in only the second adjacent blank being numbered (if the appropriate `-ha`, `-ba`, and/or `-fa` option is set). Default is `1`.

`-d`*xx* The delimiter characters specifying the start of a logical page section may be changed from the default characters (`\:`) to two user-specified characters. These characters must be single-byte characters. If only one character is entered, the second character remains the default character (:). No space should appear between the `-d` and the delimiter characters. (Because backslash is a special character to the command interpreter, use two backslashes to enter one.)

EXAMPLE

The command:

```
nl -v10 -i10 -d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(BU_CMD).

LEVEL

Level 1.

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

nohup *command* [*arguments*]

DESCRIPTION

The command nohup executes *command* with the signals SIGHUP and SIGQUIT ignored. If output is not re-directed by the user, both standard output and standard error are sent to nohup.out. If nohup.out is not writable in the current directory, output is redirected to $HOME/nohup.out.

USAGE

General.

It is frequently desirable to apply nohup to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file; this file can then be executed as *command*, and the nohup applies to everything in the file.

SEE ALSO

sh(BU_CMD), signal(BA_OS).

LEVEL

Level 1.

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

`pack` [`-`] [`-f`] *name* ...

`pcat` *name* ...

`unpack` *name* ...

DESCRIPTION

The command `pack` attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name*`.z` with the same access modes, access and modified dates, and owner as those of *name*. The option `-f` will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If `pack` is successful, *name* will be removed. Packed files can be restored to their original form using `unpack` or `pcat`.

The command `pack` uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the `-` argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of `-` in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each `.z` file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

The command `pack` returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;

The command `pcat` does for packed files what `cat` does for regular files, except that `pcat` cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*`.z` use:

```
pcat name.z
```

or

```
pcat name
```

To make an unpacked copy, called `abc`, of a packed file named *name.z* (without destroying *name.z*) use the command:

```
pcat name >abc
```

The command `pcat` returns the number of files it was unable to unpack. Failure may occur if:

the file name (exclusive of the `.z`) has more than {NAME_MAX}-2 characters;
the file cannot be opened;
the file does not appear to be the output of `pack`.

The command `unpack` expands files created by `pack`. For each file *name* specified in the command, a search is made for a file called *name*`.z` (or just *name*, if *name* ends in `.z`). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the `.z` suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

The command `unpack` returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in `pcat`, as well as for the following:

a file with the ''unpacked'' name already exists;
the unpacked file cannot be created.

SEE ALSO

cat(BU_CMD).

LEVEL

Level 1.

NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

`paste` *file1 file2 ...*
`paste -d` *list file1 file2 ...*
`paste -s` [`-d` *list*] *file1 file2 ...*

DESCRIPTION

In the first two forms, `paste` concatenates corresponding lines of the given input files *file1*, *file2*, *etc.* The file-name – means standard input. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). In the last form above (`–s` option), `paste` combines subsequent lines of the input file (serial merging).

In all cases, lines are glued together with the `tab` character, unless the `–d` option is used (see below).

Output is to the standard output, so that `paste` can be used as the start of a pipe, or as a filter, if – is used in place of a file name.

Without the `–d` option, the newline characters of each but the last file (or last line in case of the `–s` option) are replaced by a `tab` character.

When the `–d` option is used, a character from the *list* immediately following `–d` replaces the default `tab` as the line concatenation character. The *list* is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no `–s` option), the lines from the last file are always terminated with a newline character, not from the *list*. The *list* may contain the special escape sequences: `\n` (newline), `\t` (tab), `\\` (backslash), and `\0` (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the command interpreter.

EXAMPLE

```
ls | paste – – – –
```
list directory in four columns

```
paste –s –d"\t\n" file
```
combine pairs of lines into lines

SEE ALSO

cut(BU_CMD), grep(BU_CMD), pr(BU_CMD).

LEVEL

Level 1.

NAME

pfmt – display error message in standard format

SYNOPSIS

pfmt [-l *label***] [-s** *severity***] [-g** *catalog*:*msgid***]** *format* **[***args***]**

DESCRIPTION

pfmt uses *format* for **printf** style formatting of *args*. **pfmt** encapsulates the output in the standard error message format and displays it on *stderr*.

The following options are available.

-l *label* Specify the label string to be displayed with the message (e.g., **"UX:cat"**). *label* is a character string no more than 25 characters in length; it will be automatically suffixed with a colon (**:**). When unspecified, no label is displayed as part of the message.

-s *severity* Specify the severity string to be displayed with the message. Acceptable strings include the standard severities in either their print string (i.e., **HALT**, **ERROR**, **INFO**, **WARNING**, and **"TO FIX"**) or keyword (i.e., **halt**, **error**, **info**, **warn**, and **action**) forms, or any other user-defined string. The severity will be suffixed with a colon (**:**). The **ERROR** severity will be used if no severity is specified.

-g *catalog*:*msgid*

Specify that a localized version of *format* should be retrieved from a locale-specific message database. *catalog* indicates the message database that contains the localized version of the *format* string. *catalog* must be limited to 14 characters. These characters must be selected from a set of all characters values, excluding **\0** (null) and the ASCII codes for **/** (slash) and **:** (colon).

msgid is a positive number that indicates the index of the string into the message database.

If *catalog* does not exist in the current locale (identified by the **LC_MESSAGES** or **LANG** environment variables), or if *msgid* is out of bounds, **pfmt** will attempt to retrieve the message from the C locale. If this second retrieval fails, **pfmt** uses the *format* string as passed on the command line.

pfmt will output **Message not found!!\n** as the *format* string if *catalog* is not a valid catalog name as defined above and *msgid* is not a positive number.

Standard Error Message Format

pfmt displays error messages in the following format:

*label***:** *severity***:** *text*

If no *label* was defined using the **-l** *label* option, the message is displayed in the format:

*severity***:** *text*

If **pfmt** is called twice to display an error message and a helpful *action* or recovery message, the output can look like the following:

label: *severity*: *text*
label: **TO FIX:** *text*

RETURN VALUE

Upon success, **pfmt** exits with code 0.

Upon failure, **pfmt** exits with the following codes:

1 write error.

3 syntax error.

EXAMPLE

Example 1:

```
pfmt -l UX:test -s error "Syntax error"
```

displays the message:

```
UX:test: ERROR: Syntax error
```

SEE ALSO

envvar(BA_ENV), gettxt(BU_CMD), lfmt(BU_CMD), printf(BU_CMD), lfmt(BA_LIB), pfmt(BA_LIB).

LEVEL

Level 2: April 1991.

NAME

pg – file perusal filter for CRTs

SYNOPSIS

pg [*-number***] [-p** *string***] [-cefnrs] [+***linenumber***] [+/***pattern***/] [***file* **. . .]**

DESCRIPTION

The **pg** command is a filter that allows the examination of *files* one screenful at a time on a CRT. (If no *file* is specified or if it encounters the file name **-**, **pg** reads from standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are listed below. **pg** processes supplementary code set characters in *files*, and recognizes supplementary code set characters in the *string* given to the **-p** option (see below) according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar**(BA_ENV)]. In regular expressions, pattern searches are performed on characters, not bytes, as described on **ed**(BU_CMD).

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

To determine terminal attributes, **pg** scans the **terminfo** data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

–number
: An integer specifying the size (in lines) of the window that **pg** is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

-c
: Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the **terminfo** data base.

-e
: Causes **pg** not to pause at the end of each file.

-f
: Normally, **pg** splits lines longer than the screen width at characters, but some sequences of characters in the text being displayed (for example, escape sequences for underlining) generate undesirable results. The **-f** option inhibits **pg** from splitting lines.

-n
: Normally, commands must be terminated by a *newline* character. This option causes an automatic end of command as soon as a command letter is entered.

-p *string*
: Causes **pg** to use *string* as the prompt. If the prompt string contains a **%d**, the first occurrence of **%d**' in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ''**:**''. *string* may contain supplementary code set characters.

-s Causes **pg** to print all messages and prompts in standout mode (usually inverse video).

+*linenumber*
Start up at *linenumber*.

+/*pattern***/**
Start up at the first line containing the regular expression pattern.

The responses that may be typed when **pg** pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands that cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)<*newline*> or <*blank*>
This causes one page to be displayed. The address is specified in pages.

(+1) **l** With a relative address this causes **pg** to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
Simulates scrolling half a screen forward or backward.

*i***f** Skip *i* screens of text.

*i***z** Same as *newline* except that *i*, if present, becomes the new default number of lines per screenful.

The following perusal commands take no *address*.

. or **^L**
Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in **ed**(BU_CMD) are available. They must always be terminated by a *newline*, even if the –*n* option is specified.

*i***/***pattern***/**
Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i^*pattern*^
*i***?***pattern***?**
Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds

100 terminals which will not properly handle the ?.

After searching, **pg** will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of **pg** can modify the environment of perusal with the following commands:

*i***n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w** Display another window of text. If *i* is present, set the window size to *i*.

s *filename*
Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a *newline*, even if the *–n* option is specified.

h Help by displaying an abbreviated summary of available commands.

q or **Q** Quit **pg**.

!*command*
Command is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a *newline*, even if the *–n* option is specified.

At any time when output is being sent to the terminal, the user can press the quit key (normally CTRL-\) or the interrupt (break) key. This causes **pg** to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, because any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

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

EXAMPLES

The following command line uses **pg** to read the system news:

```
news | pg -p "(Page %d):"
```

SEE ALSO

ed (BU_CMD), **grep** (BU_CMD), **more** (BU_CMD),

LEVEL

Level 2, July 1992.

NOTICES

While waiting for terminal input, **pg** responds to BREAK, DEL, and CTRL-\ by terminating execution. Between prompts, however, these signals interrupt **pg**'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

The terminal **/**, **^**, or **?** may be omitted from the searching commands.

If terminal tabs are not set every eight positions, undesirable results may occur.

When using **pg** as a filter with another command that changes the terminal I/O options, terminal settings may not be restored correctly.

NAME

pr – print files

SYNOPSIS

`pr` [[`-`*column*] [`-w`*width*] [`-a`]] [`-e`[*c*]*k*] [`-i`[*c*]*k*] [`-dtrfp`] [`+`*page*] [`-n`[*c*]*k*] [`-o`*offset*] [`-l`*length*] [`-s`*separator*] [`-h`*header*] [`-F`] [*file* ...]

`pr` [[`-m`] [`-w`*width*]] [`-e`[*c*]*k*] [`-i`[*c*]*k*] [`-dtrfp`] [`+`*page*] [`-n`[*c*]*k*] [`-o`*offset*] [`-l`*length*] [`-s`*separator*] [`-h`*header*] [`-F`] *file1 file2...*

DESCRIPTION

The command `pr` prints the named files on the standard output. If *file* is `–`, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

For single column output, line width may not be set and is unlimited.

For multi-column output, by default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the `–s` option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until `pr` has completed printing.

The *options* below may appear singly, or may be combined in any order:

`+`*page* Begin printing with page *page* (default is 1).

`–`*column* Produce *column* columns of output (default is 1). This option should not be used with `–m`. The options `–e` and `–i` are assumed for multi-column output.

`–a` Print multi-column output across the page. This option is appropriate only with the `–`*column* option.

`–m` Merge and print all files simultaneously, one per column. This option should not be used with `–`*column*.

`–d` Double-space the output.

`–e`[*c*]*k* Expand input tabs to character positions *k*+1, 2**k*+1, 3**k*+1, *etc.* If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character). The tab character *c* must be a single byte character. *k* is the tab position specified in columns, not in characters.

`–i`[*c*]*k* In `output`, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, *etc.* If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character). The tab character *c* must be a single byte character. *k* is the tab position specified in columns, not in characters.

`-n`[*c*]*k* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of `-m` output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab). The tab character *c* must be a single byte character. *k* is the tab position specified in columns, not in characters.

`-w`*width* Set the width of a line to *width* character positions for output (default is 72). Note: this applies to both single and multi-column modes.

`-o`*offset* Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the *width* and *offset*.

`-l`*length* Set the length of a page to *length* lines (default is 66). If *length* is less than what is needed for the page header and trailer, then the option `-t` is in effect; that is, header and trailer lines are suppressed in order to make room for text.

`-h` *header* Use *header* as the header to be printed instead of the file name.

`-p` Pause before beginning each page if the output is directed to a terminal (`pr` will ring the bell at the terminal and wait for a carriage return).

`-f` Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

`-r` Print no diagnostic reports on failure to open files.

`-t` Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

`-s`*c* Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab). *c* must be a single byte character.

`-F` Fold the lines of the input file. When used in multi-column mode (with the `-a` or `-m` options) lines will be folded to fit the current column's width; otherwise, they will be folded to fit the current line width.

EXAMPLE

Print *file1* and *file2* as a double-spaced, three-column listing headed by ''file list'':

```
pr -3dh "file list" file1 file2
```

Write *file1* on *file2*, expanding tabs to columns 10, 19, 28, . . . :

```
pr -e9 -t <file1 >file2
```

LEVEL

Level 1.

NAME

printf – print a text string

SYNOPSIS

`printf` *format* [*arg* ...]

DESCRIPTION

The `printf` command converts, formats, and prints its arguments under control of the *format*. It fully supports conversion specifications for strings (`%s` descriptor); however, the results are undefined for the other conversion specifications [see printf(BA_LIB)].

format a character string that contains three types of objects:

1) plain characters, which are simply copied to the output stream.

2) conversion specifications, each of which results in fetching of zero or more arguments.

3) C-language escape sequences, which are translated into the corresponding characters.

arg string(s) to be printed under the control of *format*. The results are undefined if there are insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the excess arguments are simply ignored.

Each conversion specification is introduced by the character `%`. After the `%`, the following appear in sequence:

An optional field, consisting of a decimal digit string followed by a `$`, specifying the next *arg* to be converted. If this field is not provided, the *arg* following the last *arg* converted will be used.

Zero or more flags, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag –, described below, has been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.

An optional precision that gives the maximum number of characters to be printed from a string in `s` conversion. The precision takes the form of a period (`.`) followed by a decimal digit string; a null digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

A field width or precision or both may be indicated by an asterisk (`*`) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the arguments specifying field width or precision must appear before the *arg* (if any) to be converted. A negative field width argument is taken as a – flag followed by a positive field width. If the precision argument is negative, it will be changed to zero.

The conversion characters and their meanings are:

`s` The *arg* is taken to be a string and characters from the string are printed until a null character (`\0`) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A null value for *arg* will yield undefined results.

`%` Print a %; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result.

EXAMPLES

```
printf '%s %s %s\n' Good Morning World
```

results in the output:

```
Good Morning World
```

```
printf '%2$s %s %1$s\n' World Good Morning
```

produces the same output.

SEE ALSO

printf(BA_LIB).

LEVEL

Level 1.

NAME

ps – report process status

SYNOPSIS

ps [*options***]**

DESCRIPTION

ps 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 double quotes and separated by commas or spaces. Values for *proclist* and *grplist* must be numeric.

The *options* are:

-e	Print information about **e**very process now running.
-d	Print information about all processes except session leaders.
-a	Print information about **a**ll processes most frequently requested: all those except session leaders and processes not associated with a terminal.
-j	Print session ID and process group ID.
-f	Generate a **f**ull listing. (See below for significance of columns in a full listing.)
-l	Generate a **l**ong listing. (See below.)
-Z	Print the fully qualified Mandatory Access Control level at which the process is running; valid only if the Enhanced Security Utilities are installed.
-z	Print the alias of the Mandatory Access Control level at which the process is running; valid only if the Enhanced Security Utilities are installed.
-c	Print information in a format that reflects scheduler properties as described in **priocntl**(RT_CMD). The **-c** option affects the output of the **-f** and **-l** options, as described below.
-t *termlist*	List only process data associated with the terminal given in *termlist*. Terminal identifiers may be specified in one of two forms: the device's file name (for example, **term/04**) or, if the device's file name starts with **term**, just the digit identifier (for example, **04**).
-p *proclist*	List only process data whose process **ID** numbers are given in *proclist*.
-u *uidlist*	List only process data whose 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.
-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.
-s *sesslist*	List information on all session leaders whose IDs appear in *sesslist*.

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 (**f**ull or **l**ong, respectively) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that 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

- 00 Process has terminated: process table entry now available.
- 01 A system process: always in primary memory.
- 02 Parent is tracing process.
- 04 Tracing parent's signal has stopped process: parent is waiting
- 08 Process is currently in primary memory.
- 10 Process currently in primary memory: locked until an event completes.
- 20 Process cannot be swapped.

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.
- I Idle: process is being created.
- Z Zombie state: process terminated and parent not waiting.
- T Traced: process stopped by a signal because parent is tracing it.
- X SXBRK state: process is waiting for more primary memory.

UID (f,l) The user **ID** number of the process owner (the login name is printed under the **-f** option).

PID (all) The process **ID** of the process (This information is necessary to kill a process).

PPID (f,l) The process **ID** of the parent process.

C (f,l) Processor utilization for scheduling. Not printed when the **-c** option is used.

CLS (f,l) Scheduling class for the process. 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 time-sharing class have a nice value.
ADDR	(l)	The memory address of the process.
SZ	(l)	The size (in pages or clicks) of the virtual address space of the process.
WCHAN	(l)	The address of an event for which the process is sleeping, or in SXBRK state, (if blank, the process 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.
COMMAND	(all)	The command name (the full command name and its arguments are printed under the **-f** option).
LEVEL	(Z,z)	The Mandatory Access Control level (fully qualified level name or alias).

The **-z** and **-Z** options are mutually exclusive and valid only if the Enhanced Security Utilities are installed. If the **-z** option is specified and no alias is assigned to the level, the decimal value of the level identifier (LID) is displayed. If the **-Z** option is specified and the LID is in the **valid-inactive** state, the decimal value of the LID is displayed. LID states are described in **lvlname**(1M).

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Files

/dev	
/dev/sxt/*	
/dev/term/*	
/dev/xt/*	terminal (‘‘tty’’) names searcher files
/proc/*	process information
/etc/p p	

If no *termlist*, *proclist*, *uidlist*, or *grplist* is specified, **ps** checks **stdin**, **stdout**, and **stderr** in that order, looking for the controlling terminal and will attempt to report on processes associated with the controlling terminal. In this situation, if **stdin**, **stdout**, and **stderr** are all redirected, **ps** will not find a controlling terminal, so there will be no report.

ps -ef 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.

The **-x** option will not be in future releases. It has no effect unless combined with the **-l** option.

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

The command pwd prints the path name of the working (current) directory.

ERRORS

`Cannot open ...` and `Read error in ...` may indicate possible file system trouble.

SEE ALSO

cd(BU_CMD).

LEVEL

Level 1.

NAME

rm, rmdir – remove files or directories

SYNOPSIS

`rm [-fri]` *file ...*

`rmdir` *dir ...*

DESCRIPTION

The command `rm` removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory as a minimum, and in addition may require write permission of the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and the user is asked for a response. If the response begins with `y` the file is deleted, otherwise the file remains. No questions are asked when the option `-f` is given or if the standard input is not a terminal.

If *file* is a symbolic link, the link will be removed, but the file or directory to which it refers will not be deleted. A user does not need write permissions on a symbolic link to remove it, provided the user has write permissions in the directory.

If a designated file is a directory, an error comment is printed unless the optional argument `-r` has been used. In that case, `rm` recursively deletes the entire contents of the specified directory, and the directory itself. 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 command will always fail (even if the `-f` option is used), resulting in an error message.

If the option `-i` (interactive) is in used, `rm` asks whether to delete each file, and, under `-r`, whether to examine each directory.

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

ERRORS

To avoid the consequences of inadvertently doing something like:

```
rm -r .*
```

it is forbidden to remove the files `.` or `..`

SEE ALSO

unlink(BA_OS).

LEVEL

Level 1.

NAME

sed – stream editor

SYNOPSIS

`sed` [`-n`] [`-e` *script*] [`-f` *sfile*] [*files*]

DESCRIPTION

The command `sed` copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The `-f` option causes the script to be taken from file *sfile*; these options accumulate. If there is just one `-e` option and no `-f` options, the flag `-e` may be omitted. The `-n` option suppresses the default output. A *script* consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, `sed` cyclically copies a line of input into a *pattern space* (unless something is left after a `D` command), applies in sequence all commands whose addresses select that pattern space, and at the end of the script copies the pattern space to the standard output (except under `-n`) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An address is either a decimal number that counts input lines cumulatively across files, a `$` that addresses the last line of input, or a context address, *i.e.*, a `/`*regular expression*`/` in the style of the `ed` command modified as follows:

In a context address, the construction `\?`*regular expression*?, where ? is any character, is identical to `/`*regular expression*`/`. Note that in the context address `\xabc\xdefx`, the second `x` stands for itself, so that the regular expression is `abcxdef`.

The escape sequence `\n` matches a newline embedded in the pattern space.

A period `.` matches any character except the terminal newline of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function `!` (below).

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

The argument *text* consists of one or more lines, all but the last of which end with \ to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an `s` command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The argument *rfile* or the argument *wfile* must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be, at most, 10 distinct *wfile* arguments.

(1) `a\`
text Append. Place *text* on the output before reading the next input line.

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

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

(2) `d` Delete the pattern space. Start the next cycle.

(2) `D` Delete the initial segment of the pattern space through the first newline. Start the next cycle.

(2) `g` Replace the contents of the pattern space with 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 with the contents of the pattern space.

(2) `H` Append the contents of the pattern space to the hold space.

(1) `i\`
text Insert. Place *text* on the standard output.

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

(2) `n` Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

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

(2) `p` Print. Copy the pattern space to the standard output.

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

(1) `q` Quit. Branch to the end of the script. Do not start a new cycle.

(2) `r` *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) s/*regular expression*/*replacement*/*flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see ed(BU_CMD). The value of *flags* is zero or more of:

n *n*=1-512. Substitute for just the *n*th occurrence of the *regular expression*.

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

p Print the pattern space if a replacement was made.

w *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) t *label* Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a t. If *label* is empty, branch to the end of the script.

(2) w *wfile* Write. Append the pattern space to *wfile*.

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

(2) y/*string1*/*string2*/
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. *String1* and *string2* must have the same number of characters. When characters from supplementary code sets are specified for *string1* and *string2* for the y command, the results of processing cannot be guaranteed.

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

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

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

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

(0) An empty command is ignored.

(0) # If a sharp (#) appears as the first character on the first line of a script file, that entire line is treated as a comment, with one exception. If the character after the # is an 'n', 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, non-empty command line.

SEE ALSO

awk(BU_CMD), ed(BU_CMD), grep(BU_CMD).

LEVEL

Level 1.

NAME

sh, jsh, rsh – shell, the standard / restricted command interpreter

SYNOPSIS

sh [*flags*] [*args*]
jsh [*flags*] [*args*]
rsh [*flags*] [*args*]

DESCRIPTION

The command sh is an interface to a shell that interprets and executes commands read from a terminal or a file. The command jsh is an interface to the shell which provides all the functionality of sh and enables Job Control (see ''Job Control'' below). The command rsh provides an interface to a restricted version of the standard command interpreter sh; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See ''Invocation'' below for the meaning of flags and other arguments to the shell.

Definitions

A *blank* is a tab or a space.
A *name* is a sequence of ASCII letters, digits, or underscores, beginning with a letter or underscore.
A *parameter* is a name, a digit, or any of the characters *, @, #, ?, –, $, and !.

Commands

A *simple command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 [see exec(BA_OS)]. The *value* of a *simple command* is its exit status if it terminates normally, or (octal) 200+status if it terminates abnormally [see signal(BA_OS) for a list of status values].

A *pipeline* is a sequence of one or more commands separated by |. The standard output of each command except the last is connected by a ''pipe'' [see pipe(BA_OS)] 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. Note that the above definition of a *pipeline* includes the degenerate case of a single command with no pipeline operator. We refer to such a case as a *pipeline* for notational convenience.

A *list* is a sequence of one or more pipelines separated by ;, &, &&, or ||, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. The symbol ; causes sequential execution of the preceding pipeline (that is, the shell waits for the pipeline to finish before executing any commands following the semicolon); the symbol & causes asynchronous execution of the preceding pipeline (that is, the shell does not wait for that pipeline to finish). The symbol && (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of newlines may appear in a *list*, instead of semicolons, to delimit commands.

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

`for` *name* [`in` *word* . . .] `do` *list* `done`
: Each time a `for` command is executed, *name* is set to the next *word* taken from the `in` *word* list. If `in` *word* . . . is omitted, the `for` command executes the `do` *list* once for each positional parameter that is set (see ''Parameter Substitution'' below). Execution ends when no more words are in the list.

`case` *word* `in` [*pattern* [`|` *pattern*] . . .) *list* `;;`] . . . `esac`
: A `case` command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see ''Filename Generation'') except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

`if` *list* `then` *list* [`elif` *list* `then` *list*] . . . [`else` *list*] `fi`
: The *list* following `if` is executed and, if it returns a zero exit status, the *list* following the first `then` is executed. Otherwise, the *list* following `elif` is executed and, if its value is zero, the *list* following the next `then` is executed. Failing that, the `else` *list* is executed. If no `else` *list* or `then` *list* is executed, then the `if` command returns a zero exit status.

`while` *list* `do` *list* `done`
: A `while` command repeatedly executes the `while` *list* and, if the exit status of the last command in the list is zero, executes the `do` *list*; otherwise the loop terminates. If no commands in the `do` *list* are executed, then the `while` command returns a zero exit status; `until` may be used in place of `while` to negate the loop termination test.

(*list*)
: Execute *list* in a sub-shell.

{*list*;}
: *list* is simply executed. The `{` must be followed by a space. (The semicolon may be replaced by a newline.)

name() {*list*;}
: Define a function which is referenced by *name*. The body of the function is the *list* of commands between `{` and `}`. The `{` must be followed by a space. (The semicolon may be replaced by a newline.) Execution of functions is described below (see ''Execution''). The `{` and `}` (open and close braces, respectively) are not required if the body of the function is a simple command as defined above, under ''Commands.''

The following words are only recognized as the first word of a command and when not quoted:

```
if then else elif fi case esac for
while until do done { }
```

Comments

A word beginning with # causes that word and all the characters after it, up to a newline, to be ignored.

Command Substitution

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

No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (‘) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes, a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a newline character (\newline), both the backslash and the newline are removed (see ‘‘Quoting’’ below). In addition, backslashes used to escape dollar signs (\$) are removed. Since parameter substitution is not done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \, ‘, ", newline, and $ are left intact when the command string is read.

Parameter Substitution

The character $ is used to introduce substitutable parameters. There are two types of parameters, positional and keyword. If the parameter name is a single digit (0–9), it is a positional parameter; otherwise, the name must be a legal *name* as defined above, and gives a keyword parameter. Positional parameters may be assigned values by `set`. Parameter `$0` is set from argument zero when the shell is invoked. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] . . .

Pattern matching is not performed on *value*. A function and a variable cannot have the same *name*.

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

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

${*parameter*:=*word*}
: If *parameter* is not set or is null, set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned in this way.

${*parameter*:?*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message:

```
parameter null or not set
```

is printed.

${*parameter*:+*word*}
If *parameter* is set and is non-null, substitute *word*; otherwise, substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, `pwd` is executed only if `d` is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (`:`) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

`#` The number of positional parameters in decimal.

`-` Flags supplied to the shell on invocation or by the `set` command.

`?` The decimal value returned by the last synchronously executed command.

`$` The process number of this shell.

`!` The process number of the last background command invoked.

The shell uses the following parameters:

`HOME` The default argument (home directory) for the `cd` command.

`PATH` The search path for commands (see ''Execution'' below). The user may not change `PATH` if executing under `rsh`.

`CDPATH` The search path for the `cd` command. The syntax and usage is similar to that of `PATH`.

`IFS` Internal field separators, normally space, tab, and newline (see ''Blank Interpretation'' below).

`MAIL` If this parameter is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file. The user is informed only if `MAIL` is set and `MAILPATH` is not set.

`MAILCHECK`
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the `MAILPATH` or `MAIL` parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each primary prompt.

`MAILPATH` A colon (`:`) separated list of filenames. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each filename can be followed by `%` and a message that will be printed when the modification time changes. The default message is:

```
you have mail.
```

`PS1` Primary prompt string, by default "`$ `".

`PS2` Secondary prompt string, by default "`> `".

`LANG` If this parameter is set, the shell will use it to determine the current locale [see env(SD_CMD) and setlocale(BA_OS)].

`SHACCT` If this parameter is set to the name of a file writable by the user, the shell writes an accounting record in the file for each shell procedure executed.

`SHELL` When the shell is invoked, it scans the environment (see ''Environment'' below) for this name. If it is found and an `r` appears in the filename part of its value, the shell becomes a restricted shell.

The shell gives default values to `PATH`, `PS1`, `PS2`, `MAILCHECK` and `IFS`.

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in `IFS`) and are split into distinct arguments where such characters are found. Explicit null arguments (`""` or `''`) are retained. Implicit null arguments (those resulting from parameters that have no values) are removed.

Filename Generation

Following substitution, each command *word* is scanned for the characters `*`, `?`, and `[`. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If a filename that matches the pattern is not found, the word is left unchanged. The character . at the start of a filename or immediately following a `/`, as well as the character `/` itself, must be matched explicitly.

`*` Matches any string, including the null string.

`?` Matches any single character.

`[...]` Matches any one of the enclosed characters. A pair of characters separated by `-` matches any character lexically between the pair, inclusive. If the first character following the opening `[` is a `!`, any character not enclosed is matched.

Note that all quoted characters (see below) must be matched explicitly in a filename.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

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

Any character except $, \, or ' may be *quoted* (that is, made to stand for itself) by preceding it with a backslash (\) or inserting it between a pair of single or double quote marks (for example, `'*'` or `"*"`). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair \newline is removed from a word before command and parameter substitution.

With the exception of a single quote mark, all characters enclosed between a pair of single quotes (‘*‘) are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, "‘").

Inside a pair of double quote marks (""), parameter and command substitution occurs; the shell quotes the results to avoid blank interpretation and filename generation. If $* is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces ("$1 $2 . . . "). However, if $@ is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces ("$1" "$2" . . .). \ quotes the characters \, ‘, ", and $. The pair \newline is removed before parameter and command substitution. If a backslash precedes characters other than \, ‘, ", $, and newline, then the backslash itself is quoted by the shell.

Prompting

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

Input/Output

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

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

>*word* Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise, it is truncated to zero length.

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

<<[-]*word* After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file. If, however, - is appended to <<:

1) leading tabs are stripped from *word* before the shell input is read (but after parameter and command substitution is done on *word*),

2) leading tabs are stripped from the shell input as it is read and before each line is compared with *word*, and

3) shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file.

If any character of *word* is quoted (see ‘‘Quoting’’ above), no additional processing is done to the shell input. If no characters of *word* are quoted:

1) parameter and command substitution occurs,

2) (escaped) \newline pairs are removed, and

The resulting document becomes the standard input.

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

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

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

`&. . . 2>&1`

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

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

`. . . 1>`*xxx* `2>&1`

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

Using the terminology introduced on the first page, under ''Commands,'' if a *command* is composed of several *simple commands*, redirection is evaluated for the entire *command* before it is evaluated for each *simple command*. That is, the shell evaluates redirection in the following order: for the entire *list*, each *pipeline* within the *list*, each *command* within each *pipeline*, and each *list* within each *command*.

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

Redirection of output is not allowed in the restricted shell.

Environment

The *environment* is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the `export` command is used to bind the parameter of the shell to the environment (see also `set -a`). A parameter may be removed from the environment with the `unset` command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by `unset`, plus any modifications or additions, all of which must be noted in `export` commands.

The environment for any *simple command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

`TERM=450` *cmd*

and

(`export TERM; TERM=450;` *cmd*)

(where *cmd* uses the value of the environmental variable `TERM`) are equivalent as far as the execution of *cmd* is concerned if *cmd* is not a special command. If *cmd* is a Special Command, then

`TERM=450` *cmd*

modifies the `TERM` variable in the current shell.

If the `-k` flag is set, all keyword arguments are placed in the environment, even if they occur after the command name. For example, in the the following sequence, the first line prints ''`a=b c`''; the third line, ''`c`'':

```
echo a=b c
set –k
echo a=b c
```

Signals

The `INTERRUPT` and `QUIT` signals for an invoked command are ignored if the command is followed by `&`; otherwise, signals have the values inherited by the shell from its parent, with the exception of the signal `SIGSEGV` (see also the `trap` command below). Likewise, when invoked as `sh`, the shell ignores the Job Control signals `SIGTSTP`, `SIGCONT`, `SIGTTIN` and `SIGTTOU` [see signal(BA_OS)].

Execution

Each time a command is executed, the command substitution, parameter substitution, blank interpretation, input/output redirection, and filename generation listed above are carried out. If the command name matches the name of a defined function, it is executed in the shell process. If the command name does not match a defined function, but matches one of the Special Commands listed below [see ''Special Commands''], it is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters `$1`, `$2`, . . . are set to the arguments of the function. If the command name matches neither a Special Command nor the name of a defined function, a new process is created and an attempt is made to execute the command via an exec routine [see exec(BA_OS)].

The variable `PATH` defines the search path for the directory containing the command. Alternative directory names are separated by a colon (`:`). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a `/` the search path is not used; such commands are not executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an executable (`a.out`) file, it is assumed to be a file containing shell commands. A subshell is spawned to read it. A parenthesized command is also executed in a subshell.

The location in the search path where a command was found is remembered by the shell. If the command was found in a relative directory, its location must be redetermined whenever the current directory changes. The shell forgets all remembered locations whenever the `PATH` variable is changed or the `hash -r` command is executed (see below).

Special Commands

Input/output redirection is permitted for these commands. File descriptor `1` is the default output location. When Job Control is enabled additional commands are added to the shell's environment (see ''Job Control'').

`:` No effect; the command does nothing. A zero exit code is returned.

`.` *file* Read and execute commands from *file* and return. The search path specified by `PATH` is used to find the directory containing *file*.

`break` [*n*]
Exit from the enclosing `for` or `while` loop, if any. If *n* is specified, break *n* levels.

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

`cd` [*arg*]
Change the current directory to *arg*. The variable `HOME` is the default *arg*. The 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 pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a `/` the search path is not used. Otherwise, each directory in the path is searched for *arg*. The `cd` command may not be executed by `rsh`.

`echo` [*arg* . . .]
Echo arguments. [See echo(BU_CMD) for usage and description.]

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

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

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

`export` [*name* . . .]
The given *names* are marked for automatic export to the *environment* of subsequently executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names may not be exported.

`hash [-r] [`*name . . .* `]`
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The `-r` option causes the shell to forget all remembered locations. If no arguments are given, information 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.

`mldmode`

`mldmode -r` [*string*]

`mldmode -v` [*string*]
With no arguments, the current multilevel directory (MLD) mode is reported. If `-r` alone is specified, the MLD mode of the interactive shell is changed to *real* mode. If `-v` alone is specified, the MLD mode of the interactive shell is changed to *virtual* mode. If the `-r` or `-v` option is followed by a *string* specifying a command, that command alone is executed in the specified MLD mode. The default mode upon login is *virtual* mode.

`newgrp [`*arg*`]`
Equivalent to `exec newgrp` *arg*. See newgrp(AU_CMD) for usage and description.

`priv [+|-`*priv_name . . .* `]` *set_name* [. . .]
For each *set_name*, `priv` sets or displays the privileges contained in that privilege set. *set_name* may be either `max` for the maximum privilege set or `work` for the working set. *priv_name* is the name of a privilege. If *priv_names* are supplied, `priv` scans the list and turns off thoses privileges that are preceded by a minus sign and turns on those that are preceded by a plus sign in each of the sets listed. If *priv_names* are not supplied, the `priv` command prints the current list of privileges for each of the requested sets.

`pwd` Print the current working directory. [See pwd(BU_CMD) for usage and description.]

`read` *name1* [*name2 . . .*]
One line is read from the standard input and, using the internal field separator, `IFS` (normally space or tab), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, and so on, with leftover words assigned to the last *name*. Lines can be continued using \newline. Characters other than newline can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is `0`, unless an end-of-file is encountered.

`readonly` [*name . . .*]
The given *names* are marked `readonly` and the values of these *name*(*s*) may not be changed by subsequent assignment. If arguments are not given, a list of all `readonly` names is printed.

`return` [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

`set` [`--aefhkntuvx` [*arg* . . .]]

`--` Do not change any of the flags; useful in setting `$1` to `-`.

`-a` Mark variables which are modified or created for export.

`-e` Exit immediately if a command exits with a non-zero exit status.

`-f` Disable filename generation.

`-h` Locate and remember function commands as functions are defined. (Function commands are normally located when the function is executed.)

`-k` All keyword arguments are placed in the environment for a command, not just those that precede the command name.

`-n` Read commands but do not execute them.

`-t` Exit after reading and executing one command.

`-u` Treat unset variables as an error when substituting.

`-v` Print shell input lines as they are read.

`-x` Print commands and their arguments as they are executed.

Using `+` rather than `-` causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in `$-`. The remaining arguments are positional parameters and are assigned, in order, to `$1`, `$2`, ... If arguments are not given, the values of all names are printed.

`shift` [*n*]
The positional parameters from `$`*n*`+1` . . . are renamed `$1` . . . . If *n* is not given, it is assumed to be `1`.

`test` Evaluate conditional expressions [see test(BU_CMD) for usage and description].

`times`
Print the accumulated user and system times for processes run from the shell.

`trap` [*arg*] [*n*] . . .
The command *arg* is to be read and executed when the shell receives numeric or symbolic signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number or corresponding symbolic names. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on the signal `SIGSEGV` (memory fault) produces an error. If *arg* is absent, all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *n* is `0`, the command *arg* is executed on exit from the shell. The `trap` command with no arguments prints a list of commands associated

with each signal number.

`type` [*name . . .*]
For each *name*, indicate how it would be interpreted if used as a command name.

`ulimit` [-[HS][a|cdfnstv]]

`ulimit` [-[HS][c|d|f|n|s|t|v]] *limit*
`ulimit` prints or sets hard or soft resource limits. These limits are described in getrlimit(BA_OS).

If *limit* is not present, `ulimit` prints the specified limits. Any number of limits may be printed at one time. The `-a` option prints all limits.

If *limit* is present, `ulimit` sets the specified limit to *limit* . The string `unlimited` requests the largest valid limit. Limits may be set for only one resource at a time. Any user may set a soft limit to any value below the hard limit. Any user may lower a hard limit. Only a privileged user may raise a hard limit [see su(AU_CMD)].

The `-H` option specifies a hard limit. The `-S` option specifies a soft limit. If neither option is specified, `ulimit` will set both limits and print the soft limit.

The following options specify the resource whose limits are to be printed or set. If no option is specified, the file size limit is printed or set.

`-c` maximum core file size (in 512-byte blocks)

`-d` maximum size of data segment or heap (in kbytes)

`-f` maximum file size (in 512-byte blocks)

`-n` maximum file descriptor plus 1

`-s` maximum size of stack segment (in kbytes)

`-t` maximum CPU time (in seconds)

`-v` maximum size of virtual memory (in kbytes)

`umask` [*nnn*]
The user file-creation mask is set to *nnn* [see umask(BA_OS)]. If *nnn* is omitted, the current value of the mask is printed.

`unset` [*name . . .*]
For each *name*, remove the corresponding variable or function. The variables `PATH`, `PS1`, `PS2`, `MAILCHECK` and `IFS` cannot be unset.

`wait` [*n*]
Wait for the specified process and report its termination status. If *n* is not given, all currently active child processes are waited for and the return code is `0`.

Invocation

If the shell is invoked through an exec routine [see exec(BA_OS)] and the first character of argument zero is `-`, commands are initially read from `/etc/profile` and from `$HOME/.profile`, if such files exist. Thereafter, commands are read as described below. The flags below are interpreted by the shell on invocation only;

note that unless the `-c` or `-s` flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

`-c` *string* If the `-c` flag is present, commands are read from *string*.

`-s` If the `-s` flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for Special Commands) is written to file descriptor `2`.

`-i` If the `-i` flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case, `TERMINATE` is ignored (so that `kill 0` does not kill an interactive shell) and `INTERRUPT` is caught and ignored (so that `wait` is interruptible). In all cases, `QUIT` is ignored by the shell.

`-p` If the `-p` flag is present, the shell will not set the effective user and group IDs to the real user and group IDs.

`-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.

Job Control (jsh) Only

When the shell is invoked as `jsh`, Job Control is enabled in addition to all the functionality described previously for `sh`. Typically, Job Control is enabled for the login shell only. Subshells and non-interactive shells typically do not benefit from the added functionality of Job Control.

With Job Control enabled, every command or pipeline the user enters at the terminal is called a *job*. All jobs exist in one of the following states: foreground, background, or stopped. A job in the foreground has read and write access to the controlling terminal. A job in the background only has conditional write access to the controlling terminal [see stty(AU_CMD)]. A stopped job is one that has been placed in a suspended state, usually as a result of a `SIGTSTP` signal [see signal(BA_ENV)].

Every job the shell starts is assigned a positive integer, called a *job number*, which is tracked by the shell and is used as an identifier to indicate a specific job. Additionally the shell keeps track of the *current* and *previous* jobs. The *current job* is the most recent job to be started or restarted. The *previous job* is the first non-current job.

The acceptable syntax for a Job Identifier is of the form:

%jobid

where, *jobid* may be specified in any of the following formats:

% or + for the current job

- for the previous job

?<*string*> specify the job for which the command line uniquely contains *string*

n for job number *n*, where *n* is a job number

pref where *pref* is a unique prefix of the command name (for example, if the command `ls -l foo` were running in the background, it could be referred to as `%ls`); *pref* cannot contain blanks unless it is quoted.

When Job Control is enabled, the following commands are added to the users environment to manipulate jobs:

`bg [`

The restrictions above are enforced after `.profile` is interpreted.

A restricted shell can be invoked in one of the following ways:

- `rsh` is the filename part of the last entry in the `/etc/passwd` file [see passwd(AU_CMD)];
- the environment variable `SHELL` exists and `rsh` is the filename part of its value;
- the shell is invoked and `rsh` is the filename part of argument `0`;
- the shell is invoked with the `-r` option.

When a command to be executed is found to be a shell procedure, `rsh` invokes `sh` to execute it. Thus it is possible to provide the end-user with 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 (for example, `/usr/rbin`) that can be safely invoked by a restricted shell. Some systems also provide a restricted editor, `red`.

NOTES

Words used for filenames in input/output redirection are not interpreted for filename generation (see ''Filename Generation'' above). For example, `cat file1 >a*` will create a file named `a*`.

Because commands in pipelines are run as separate processes, variables set in a pipeline do not have an effect on the parent shell.

If you get the error message

```
cannot fork, too many processes
```

try using the `wait` command [see wait(BA_OS)] to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process IDs associated with your login, and to the number the system can keep track of.)

Only the last process in a pipeline can be waited for.

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to `exec` the original command. Use the `hash` command to correct this situation.

RETURN VALUE

Errors detected by the shell, such as syntax errors, cause the shell to return a nonzero exit status. If the shell is being used non-interactively, execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see the `exit` command above).

jsh Only

If the shell is invoked as `jsh` and an attempt is made to exit the shell while there are stopped jobs, the shell issues one warning:

```
There are stopped jobs.
```

This is the only message. If another exit attempt is made and there are still stopped jobs, `SIGHUP` and `SIGCONT` signals are sent to these jobs from the kernel and the shell is exited.

FILES

```
/dev/null
/etc/profile
/tmp/sh*
$HOME/.profile
```

SEE ALSO

cd(BU_CMD), echo(BU_CMD), env(SD_CMD), exec(BA_OS), getrlimit(BA_OS), kill(BU_CMD), newgrp(AU_CMD), passwd(AU_CMD), pipe(BA_OS), pwd(BU_CMD), setlocale(BA_OS), signal(BA_ENV), signal(BA_OS), stty(AU_CMD), su(AU_CMD), test(BU_CMD), ulimit(BA_OS), umask(BA_OS), umask(BU_CMD), wait(BA_OS).

LEVEL

Level 1.

NAME

sleep – suspend execution for an interval

SYNOPSIS

`sleep` *time*

DESCRIPTION

The command `sleep` suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

SEE ALSO

alarm(BA_OS), sleep(BA_OS).

LEVEL

Level 1.

NAME

sort – sort and / or merge files

SYNOPSIS

`sort` [`-cmu`] [`-o`*output*] [`-y`[*kmem*]] [`-z`*recsz*] [`-dfinr`] [`-btx`]
[+*pos1* [-*pos2*]] [*files*]

DESCRIPTION

The command `sort` sorts lines of all the named files together and writes the result on the standard output. The standard input is read if `-` is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The following options alter the default behavior:

`-c` Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

`-m` Merge only, the input files are already sorted.

`-u` Unique: suppress all but one in each set of lines having equal keys.

`-o`*output* The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between `-o` and *output*.

`-y`[*kmem*] The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, `sort` begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, `sort` will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, `-y0` is guaranteed to start with minimum memory. By convention, `-y` (with no argument) starts with maximum memory.

`-z`*recsz* The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the `-c` or `-m` options, a popular system default size will be used. Lines longer than the buffer size will cause `sort` to terminate abnormally. Supplying *recsz*, which is the actual number of bytes in the longest line to be merged (or some larger value), will prevent abnormal termination.

The following options override the default ordering rules.

`-d` ''Dictionary'' order: only letters, digits and blanks (spaces and tabs) are significant in comparisons. No comparison is performed for multibyte characters.

`-f` Fold lower case letters into upper case. Only applies to single byte characters.

`-i` Ignore characters not defined as printable and multibyte characters in non-numeric comparisons.

`-n` An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The `-n` option implies the `-b` option (see below). Note that the `-b` option is only effective when restricted sort key specifications are in effect.

`-r` Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation *+pos1 -pos2* restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1)*. A missing *-pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a newline. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

`-t`*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (i.e., *xx* delimits an empty field).

`-b` Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the `-b` option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the `b` flag may be attached independently to each *+pos1* or *-pos2* argument (see below).

The arguments *pos1* and *pos2* each have the form *m*`.`*n* optionally followed by one or more of the flags `bdfinr`. A starting position specified by *+m*`.`*n* is interpreted to mean the *n*+1st column in the *m*+1st field. A missing `.`*n* means `.0`, indicating the first column 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 column in the *m*+1st field.

A last position specified by *-m*`.`*n* is interpreted to mean the *n*th column (including separators) after the last character of the *m*th field. A missing `.`*n* means `.0`, indicating the last column of the *m*th field. If the `b` flag is in effect *n* is counted from the last leading blank in the *m*+1st field; *-m*`.1b` refers to the first non-blank column in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

Characters from supplementary code sets are collated in code order (EUC representation).

ERRORS

The `sort` command comments and exits with non-zero status for various trouble conditions (e.g., when input lines are too long), and for disorder discovered under the `-c` option.

When the last line of an input file is missing a newline character, `sort` appends one, prints a warning message, and continues.

EXAMPLE

Sort the contents of `infile` with the second field as the sort key:

```
sort +1 -2 infile
```

Sort, in reverse order, the contents of `infile1` and `infile2`, placing the output in `outfile` and using the first column of the second field as the sort key:

```
sort -r -o outfile +1.0 -1.2 infile1 infile2
```

Sort, in reverse order, the contents of `infile1` and `infile2` using the first non-blank column of the second field as the sort key:

```
sort -r +1.0b -1.1b infile1 infile2
```

Print the password file sorted by the numeric user ID (the third colon-separated field):

```
sort -t: +2n -3 /etc/passwd
```

Print the lines of the already sorted file `infile`, suppressing all but the first occurrence of lines having the same third field (the options `-um` with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort -um +2 -3 infile
```

SEE ALSO

comm(BU_CMD), join(AU_CMD), uniq(BU_CMD).

LEVEL

Level 1.

NAME

spell, **hashmake**, **spellin**, **hashcheck**, **compress** – find spelling errors

SYNOPSIS

spell [-v] [-b] [-x] [+*local_file***] [***files***]**

DESCRIPTION

spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and / or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

spell ignores most **troff**, **tbl**, and **eqn** constructions. It also ignores punctuation marks and special characters (for example, _ and =).

-v All words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

-b British spelling is checked. Besides preferring **centre**, **colour**, **programme**, **speciality**, **travelled**, and so on, this option insists upon *-ise* in words like **standardise**, Fowler and the OED (Oxford English Dictionary) to the contrary notwithstanding.

-x Every plausible stem is displayed, one per line, with **=** preceding each word.

+*local_file* Words found in *local_file* are removed from **spell**'s output. *local_file* is the name of a user-provided file that contains a sorted list of words, one per line. The list must be sorted with the ordering used by **sort**(BU_CMD) (for example, upper case preceding lower case). If this ordering is not followed, some entries in *local_file* may be ignored. With this option, the user can specify a set of words that are correct spellings (in addition to **spell**'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Alternate auxiliary files (spelling lists, stop list, history file) may be specified on the command line by using environment variables. These variables and their default settings are shown in the FILES section. Copies of all misspellings and entries that specify the login, tty, and time of each invocation of spell are accumulated in the *history* file. The *stop list* filters out misspellings (for example, **thier=thy-y+ier**) that would otherwise pass.

NOTICES

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions. Typically, these are kept in a separate local file that is added to the hashed *spelling_list* via **spellin**.

SEE ALSO

sed (BU_CMD), **sort** (BU_CMD), **tee** (BU_CMD),

LEVEL

Level 1.

NAME

split – split a file into pieces

SYNOPSIS

`split` [*–n*] [*file* [*name*]]

DESCRIPTION

The command `split` reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with `aa` appended, and so on lexicographically, up to `zz` (a maximum of 676 files). The argument *name* cannot be longer than `{NAME_MAX}-2` characters. If no output name is given, `x` is default.

If no input file is given, or if – is given in its stead, then the standard input is used.

SEE ALSO

csplit(AU_CMD).

LEVEL

Level 1.

NAME

strchg, strconf – change or query stream configuration

SYNOPSIS

`strchg -h` *module1*[`,` *module2* ...]
`strchg -p` [`-a`|`-u` *module*]
`strchg -f` *file*
`strconf` [`-t`|`-m` *module*]

DESCRIPTION

These commands are used to alter or query the configuration of the stream associated with the user's standard input. The `strchg` command pushes modules on and/or pops modules off the stream. The `strconf` command queries the configuration of the stream. Only the super-user or owner of a STREAMS device may alter the configuration of that stream.

With the `-h` option, `strchg` pushes (pusHes) modules onto a stream; it takes as arguments the names of one or more pushable streams modules. These modules are pushed in order; that is, *module1* is pushed first, *module2* is pushed second, *etc.*

The `-p` option pops (poPs) modules off the stream. With the `-p` option alone, `strchg` pops the topmost module from the stream. With the `-p` and `-a` options, all the modules above the topmost driver are popped. When the `-p` option is followed by `-u` *module*, then all modules above but not including *module* are popped off the stream. The `-a` and `-u` options are mutually exclusive.

With the `-f` option, the user can specify a *file* that contains a list of modules representing the desired configuration of the stream. Each module name must appear on a separate line. The first name represents the topmost module and the last name represents the module that should be closest to the driver. The `strchg` command will determine the current configuration of the stream and pop and push the necessary modules in order to end up with the desired configuration.

The `-h`, `-f` and `-p` options are mutually exclusive.

Invoked without any arguments, `strconf` prints a list of all the modules in the stream as well as the topmost driver. The list is printed with one name per line where the first name printed is the topmost module on the stream (if one exists) and the last item printed is the name of the driver. With the `-t` option, only the topmost module (if one exists) is printed. The `-m` option determines if the named *module* is present on a stream. If it is, `strconf` prints the message `yes` and returns zero. If not, `strconf` prints the message `no` and returns a non-zero value. The `-t` and `-m` options are mutually exclusive.

RETURN VALUES

`strchg` returns zero on success. It prints an error message and returns non-zero status for various error conditions, including usage error, bad module name, too many modules to push, failure of an `ioctl()` on the stream, or failure to open *file* from the `-f` option.

`Strconf` returns zero on success (for the `-m` or `-t` option, "success" means the named or topmost module is present). It returns a non-zero status if invoked with the `-m` or `-t` option and the module is not present. It prints an error message and returns non-zero status for various error conditions, including usage error or failure of an `ioctl()` on the stream.

ERRORS

If modules are pushed in the wrong order, one could end up with a stream that does not function as expected. For ttys, if the line discipline module is not pushed in the correct place, one could have a terminal that does not respond to any commands.

USAGE

General.

Only the owner of a stream and the super-user may use `strchg` to alter the configuration of that stream. Users with read permissions on a stream (and the super-user) may use `strconf` to print the configuration of that stream.

EXAMPLES

The following command pushes the module `ldterm` on the stream associated with the user's standard input:

```
strchg –h ldterm
```

The following command pops the topmost module from the stream associated with `/dev/term/24`. The user must be the owner of this device or the super-user.

```
strchg –p < /dev/term/24
```

If the file `fileconf` contains the following:

```
compat
ldterm
ptem
```

then the command

```
strchg –f fileconf
```

will configure the user's standard input stream so that the module `ptem` is pushed over the driver, followed by `ldterm` and `compat` closest to the stream head.

The `strconf` command with no arguments lists the modules and topmost driver on the stream; for a stream that has only the module `ldterm` pushed above the `ports` driver, it would produce the following output:

```
ldterm
ports
```

The following command asks if `ldterm` is on the stream

```
strconf –m ldterm
```

and produces the following output while returning an exit status of 0:

```
yes
```

SEE ALSO

ioctl(BA_OS), streams(BA_DEV).

LEVEL

Level 1.

NAME

sum – print checksum and block count of a file

SYNOPSIS

`sum [-r]` *file*

DESCRIPTION

The command `sum` calculates and prints a checksum for the named file, and also prints the space used by the file, in 512-byte units. The option `–r` causes an alternate algorithm to be used in computing the checksum.

The algorithms used are uniform across all System V implementations, so that the same checksum is obtained for the same file, independent of the hardware and implementation.

SEE ALSO

wc(BU_CMD).

LEVEL

Level 1.

NAME

tail – deliver the last part of a file

SYNOPSIS

`tail` [±[*number*][`l|b|c`] [`f`]] [*file*]

`tail` [±[*number*][`l`][`r`]] [*file*]

DESCRIPTION

The command `tail` copies the named file to the standard output beginning at a designated place. If no *file* is named, the standard input is used.

Copying begins at distance +*number* from the beginning (where +1 signifies the first line of the file), or –*number* from the end of the input (if *number* is null, the value 10 is assumed). The argument *number* is counted in units of lines, blocks, or characters, according to the appended option `l`, `b`, or `c`. When no units are specified, counting is by lines.

Characters from supplementary code sets may not be displayed correctly when options `–b` or `–c` are specified, as they are processed byte-by-byte.

If the input file is not a pipe and the `–f` (‘‘follow’’) option is used, the program does not terminate after a line from the input file has been copied. Instead, `tail` enters an endless loop, i.e., `tail` sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process.

The `–r` option copies lines from the end of the file in reverse order. The default for `-r` is to print the entire file in reverse order. *number*is the count of lines from the end of the file regardless of sign.

The -r option may not be used with `–b`, `–c`, or `–f`.

USAGE

General.

`tail`s relative to the end of the file are saved in a buffer, and thus are limited in length.

EXAMPLE

The command:

```
tail –f fred
```

will print the last ten lines of the file `fred`, followed by any lines that are appended to `fred` between the time `tail` is initiated and killed. As another example, the command:

```
tail –15cf fred
```

will print the last 15 bytes of the file `fred`, followed by any lines that are appended to `fred` between the time `tail` is initiated and killed.

LEVEL

Level 1.

NAME

tee – join pipes and make copies of input

SYNOPSIS

`tee` [`-i`] [`-a`] [*file*] ...

DESCRIPTION

The command `tee` transcribes the standard input to the standard output and makes copies in the *files*. The `-i` option ignores interrupts; the `-a` option causes the output to be appended to the *files* rather than overwriting them.

LEVEL

Level 1.

NAME

test – condition evaluation command

SYNOPSIS

`test` *expr*

`[` *expr* `]`

DESCRIPTION

The command `test` evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; `test` also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr:*

`-r` *file*	true if *file* exists and is readable.
`-w` *file*	true if *file* exists and is writable.
`-x` *file*	true if *file* exists and is executable.
`-f` *file*	true if *file* exists and is a regular file.
`-d` *file*	true if *file* exists and is a directory.
`-h` *file*	true if *file* exists and is a symbolic link. With all other primitives, the symbolic links are followed by default.
`-c` *file*	true if *file* exists and is a character special file.
`-b` *file*	true if *file* exists and is a block special file.
`-p` *file*	true if *file* exists and is a named pipe (fifo).
`-u` *file*	true if *file* exists and its set-user-ID bit is set.
`-g` *file*	true if *file* exists and its set-group-ID bit is set.
`-s` *file*	true if *file* exists and has a size greater than zero.
`-t[`*fildes]*	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
`-z` *s1*	true if the length of string *s1* is zero.
`-n` *s1*	true if the length of the string *s1* is non-zero.
s1 `=` *s2*	true if strings *s1* and *s2* are identical.
s1 `!=` *s2*	true if strings *s1* and *s2* are `not` identical.
s1	true if *s1* is `not` the null string.
n1 `-eq` *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons `-ne`, `-gt`, `-ge`, `-lt`, and `-le` may be used in place of `-eq`.

These primaries may be combined with the following operators:

`!`	unary negation operator.
`-a`	binary AND operator.

`-o` binary OR operator (`-a` has higher precedence than `-o`).

`(` *expr* `)`
parentheses for grouping.

Notice that all the operators and flags are separate arguments to `test`. Notice also that parentheses are meaningful to `sh` and, therefore, must be escaped.

In the second form of the command (i.e., the one that uses `[ ]`, rather than the word `test`), the brackets must be delimited by white space.

SEE ALSO

find(BU_CMD), sh(BU_CMD).

LEVEL

Level 1.

NAME

touch – update access and modification times of a file

SYNOPSIS

`touch` [`-amc`] [*mmddhhmm*[*yy*]] [*file* ...]

DESCRIPTION

The command `touch` causes the access and modification times of each *file* to be updated. The *file* is created if it does not exist. If no time is specified the current time is used. The `-a` and `-m` options cause touch to update only the access or modification times respectively (default is `-am`). The `-c` option silently prevents `touch` from creating the *file* if it does not exist.

RETURN VALUE

The return code from `touch` is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

LEVEL

Level 1.

NAME

tr – translate characters

SYNOPSIS

`tr` [`-cds`] [*string1* [*string2*]]

DESCRIPTION

The command `tr` copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Characters specified are searched for and translated in character units, not bytes. Any combination of the options `-cds` may be used:

`-c` Complements the set of characters in *string1* with respect to the universe of characters whose octal codes are 001 through 377.

`-d` Deletes all input characters in *string1*.

`-s` Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[*a*`-`*z*] Stands for the string of characters whose octal codes run from character *a* to character *z*, inclusive. The semantics of the "`[a-z]`" notation takes after the range specification of the regular expression syntax.

[*a*`*`*n*] Stands for *n* repetitions of *a*. If the first digit of *n* is `0`, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character `\` may be used to remove special meaning from any character in a string. In addition, `\` followed by 1, 2, or 3 octal digits stands for the character whose octal code is given by those digits.

The following example creates a list of all the words in *file1*, one per line in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the command interpreter; 012 is the ASCII code for newline.

```
tr –cs "[A–Z][a–z]" "[\012*]" <file1 >file2
```

USAGE

General.

The command `tr` does not handle ASCII NUL in *string1* or *string2*; it always deletes NUL from input.

When octal notation with the backslash (\) escape character is used, a backslash is placed before each byte of multibyte characters.

LEVEL

Level 1.

NAME

true, false – provide truth values

SYNOPSIS

```
true
```

```
false
```

DESCRIPTION

The command `true` does nothing, and returns exit code zero. The command `false` does nothing, and returns a non-zero exit code. They are typically used to construct command procedures. For example,

```
while true
do
    command
done
```

SEE ALSO

sh(BU_CMD).

LEVEL

Level 1.

NAME

umask – set file-creation mode mask

SYNOPSIS

umask [*ooo***]**

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for owner, group, and others, respectively [see chmod(BU_CMD)]. The value of each specified digit is subtracted from the corresponding ''digit'' specified by the system for the creation of a file. For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

SEE ALSO

chmod(BU_CMD).

LEVEL

Level 1.

NAME

uname – print name of current system

SYNOPSIS

`uname [-snrvma]`

DESCRIPTION

The command `uname` prints the current system name on the standard output file. The options cause selected information returned by the function `uname()` [see uname(BA_OS)] to be printed:

- `-s` print the name of the implementation of the operating system
- `-n` print the nodename. The nodename may be a name by which the system is known to a communications network.
- `-r` print the operating system release.
- `-v` print the operating system version.
- `-m` print the machine hardware name.
- `-a` print all the above information.

SEE ALSO

uname(BA_OS).

LEVEL

Level 1.

NAME

uniq – report repeated lines in a file

SYNOPSIS

`uniq` [`-udc` [`+`*n*] [`-`*n*]] [*input* [*output*]]

DESCRIPTION

The command `uniq` reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. The arguments *input* and *output* should always be different. Note that repeated lines must be adjacent to be found [see sort(BU_CMD)]. If the `-u` flag is used, just the lines that are not repeated in the original file are output. The `-d` option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the `-u` and `-d` mode outputs.

The `-c` option supersedes `-u` and `-d` and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

`-`*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

`+`*n* The first *n* columns are ignored.

SEE ALSO

comm(BU_CMD), sort(BU_CMD).

LEVEL

Level 1.

NAME

wait – await completion of process

SYNOPSIS

`wait` [*pid* ...]

DESCRIPTION

With no argument, `wait` waits until all processes started with & have completed, and reports on abnormal terminations. If a numeric argument *pid* is given, and is the process ID of a background process, then `wait` waits until that process has completed.

SEE ALSO

sh(BU_CMD), wait(BA_OS).

LEVEL

Level 1.

NAME

wc – word count

SYNOPSIS

`wc` [`-lwc`] [*files*]

DESCRIPTION

The command `wc` counts lines, words, and characters in the named files, or in the standard input if no *files* appear. It also keeps a total count for all named files. A word is defined as a maximal string of characters delimited by spaces, tabs, or newlines.

The options `l`, `w`, and `c` may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is `-lwc`.

With the `-c` option, characters from supplementary code sets are counted in bytes, not characters. With `-w` option, characters from supplementary code set are ignored during counting.

When *files* are specified on the command line, their names will be printed along with the counts.

LEVEL

Level 1.

Advanced Utilities Introduction

The Advanced Utilities Extension is intended to be the next expansion step after the Basic Utilities Extension.

The following are prerequisite for support of the Advanced Utilities Extension:

- Base System
- Base Utilities Extension

Summary of Commands and Utilities

The following commands and utilities are supported by the Advanced Utilities Extension. Items marked with a (*) are Level 2, as defined in the *General Introduction* to this volume. Items marked with a ( ) are internationalized and may reference environment variables for localization information. [See envvar(BA_ENV)]. Items marked with a (†) are new to this issue of the SVID. Only those pages reflecting technical content changes or which are new to the SVID are contained in this volume.

at‡	**cu**‡	**lp**‡	**su**‡	**uustat**
atq‡	**dd**‡	**lpstat**‡	**tabs**‡	**uuto**
atrm‡	**dircmp**‡	**mailx**‡	**tar**‡	**uux**
batch‡	**ex**‡	**mesg**‡	**tty**‡	**vi**‡
cancel‡	**gencat** ‡	**newgrp**‡	**uucp**	**wall**‡
chgrp‡	**groups**	**news**‡	**uudecode**	**who**‡
chown‡	**iconv** ‡	**od**‡	**uuencode**	**write**‡
cron‡	**id**	**passwd**‡	**uulog**	
crontab‡	**join**‡	**priocntl**†	**uuname**	
csplit‡	**logname**	**stty**‡	**uupick**	

Organization of Technical Information

The ‘‘Advanced Commands and Utilities’’ chapter provides manual page descriptions of commands and utilities supported by this extension.

Advanced Commands And Utilities

The following section contains the manual pages for the AU_CMD routines.

FINAL COPY
June 15, 1995
File: au_cmd.cov
svid

NAME

at, batch – execute commands at a later time

SYNOPSIS

```
at [-f script] [-m] time [date] [+increment]
at -l [job ...]
at -r job ...
at -d job ...
at -z job ...
at -Z job ...

batch
```

DESCRIPTION

The commands `at` and `batch` read commands from standard input to be executed at a later time. The command `at` allows you to specify when the commands should be executed, while jobs queued with `batch` will execute when system load level permits.

The `-f` option reads commands to be executed from the named *script* file.

The `-m` option sends mail after the job has been run. Mail is sent even if the job does not produce output. Standard output and standard error output are mailed to the user unless redirected elsewhere.
If the Enhanced Security Extension is implemented, mail is sent at the Mandatory Access Control level of the at/batch job. `-m` has no effect on jobs which already print to standard output or standard error; the mail message will not be duplicated.

The `-l` option reports all jobs scheduled for the invoking user. When invoked by a privileged-user, all jobs scheduled are reported.

The `-r` option removes jobs previously scheduled with `at` for the invoking user. When invoked by a privileged-user, any *job* previously scheduled with `at` is removed.

The `-d` option displays the contents of the *job* specified.

The `-z` option displays the alias name of the level of the *job* specified. This option is only valid if the security extension is implemented.

The `-Z` option displays the fully qualified level of the *job* specified. This option is only valid if the security extension is implemented.

A normal user is restricted to display only information on jobs which the user owns and which are at the user's level. If a user's process is in MLD real mode [see `mldmode`(ES_CMD)], the user can display information on jobs which the user owns and are at a level that is dominated (meaning equal to or greater than) by the user's current level. An administrator is able to display information on all jobs.

The `-z` and `-Z` options are mutually exclusive. If the `-z` option is specified and there is not an alias assigned to the level, the decimal value of the level identifier (LID) is displayed. If the `-z` or `-Z` option is specified and the level is in the *valid-inactive* state, the decimal value of the LID is displayed. LID states are described in `lvlname`(ES_CMD).

The environment variables, current directory, umask, and ulimit are retained when the commands are executed. Open file descriptors, traps, and priority are lost.

Users are permitted to use `at` if their name appears in the file `/etc/cron.d/at.allow`. If that file does not exist, the file `/etc/cron.d/at.deny` is checked to determine if the user should be denied access to `at`. If neither file exists, only a user with appropriate privileges is allowed to submit a job. If only `at.deny` exists and is empty, global usage is permitted. The allow / deny files consist of one user name per line. Modification of these files should only be permitted to a privileged user.

The *time* may be specified as 1, 2, or 4 digits. One and two digit numbers are interpreted as hours, four digits as hours and minutes. The *time* may alternately be specified as two numbers separated by a colon, meaning `hour:minute`. A suffix `am` or `pm` may be appended; otherwise, a 24-hour clock time is understood. The suffix `zulu` may be used to indicate UTC. The special names `noon`, `midnight`, `now`, and `next` are also recognized.

An optional `date` may be specified as either a month name followed by a day number (and possibly year number preceded by a comma) or a day of the week (fully spelled or abbreviated to three characters). Two special ''days'', `today` and `tomorrow` are recognized. If no *date* is given, `today` is assumed if the given hour is greater than the current hour and `tomorrow` is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed.

The optional *increment* is simply a number suffixed by one of the following: `minutes`, `hours`, `days`, `weeks`, `months`, or `years`. (The singular form is also accepted.)

Thus legitimate commands include:

```
at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at 5 pm Friday
```

The commands `at` and `batch` write the job number and schedule time to standard error.

The command `batch` submits a batch job. It is almost equivalent to ''at now'', but not quite. For one, it goes into a different queue. For another, ''at now'' does not work; it is too late (and results in an error message).

The option `-r` removes jobs previously scheduled by `at` or `batch`. The *job* number is the number reported at invocation by `at` or `batch`. Job numbers can also be obtained by using the `-l` option. Only a user with appropriate privileges is allowed to remove another user's jobs.

If the environment variable `DATEMSK` is set, the `at` command will use it to interpret the user supplied date / time specification. The environment variable `DATEMSK` is set to the full pathname of the template file used to parse and interpret the user supplied date / time specification. The special names and the *increment* argument discussed previously are not recognized in this case.

The template file named by the DATEMSK environment variable should contain format lines as defined for the `date +`*format* command [see `getdate`(BA_LIB)]. The following example shows the possible contents of a template file `AT.TEMPL` in `/var/tmp`.

```
%I %p, the %est of %B of the year %Y run the following job
%I %p, the %end of %B of the year %Y run the following job
%I %p, the %erd of %B of the year %Y run the following job
%I %p, the %eth of %B of the year %Y run the following job
%d/%m/%y
%H:%M:%S
%I:%M%p
```

The following are examples of valid invocations if the environment variable DATEMSK is set to `/var/tmp/AT.TEMPL`.

```
at 2 PM, the 3rd of July of the year 2000 run the following job
at 3/4/99
at 10:30:30
at 2:30PM
```

FILES

`/etc/cron.d`	main cron directory
`/etc/cron.d/at.allow`	list of allowed users
`/etc/cron.d/at.deny`	list of denied users
`/etc/cron.d/queuedefs`	scheduling information
`/var/spool/cron/atjobs`	spool area

USAGE

The `-m` and `-f` *script* options can only be used with the job-queuing form of this command, and are therefore incompatible with the `-l` and `-r` options.

Regardless of whether the `-m` option is used, mail messages produced by `at` include a normal header with a `Subject:` line to identify the job that produced the mail.

EXAMPLE

The `at` and `batch` commands read from standard input the commands to be executed at a later time. It may be useful to redirect standard output within the specified commands.

This sequence can be used at a terminal:

```
batch
spell filename > outfile
EOT
```

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a command procedure (the sequence of output redirection specifications is significant):

```
batch <<!
spell filename 2>&1 > outfile | mail loginid
```

To have a job reschedule itself, `at` can be invoked from within the procedure.

SEE ALSO

cron(AU_CMD), date(BU_CMD).

LEVEL

Level 1.

NAME

atq – display the queue of jobs to be run at specified times

SYNOPSIS

`atq` [`-c`] [*username* ...]

`atq` [`-n`] [*username* ...]

DESCRIPTION

`atq` prints the queue of jobs, created with the `at` command, that are waiting to be run at a later date.

With no flags, the queue is sorted in chronological order of execution.

If the invoking user is a privileged-user, and no *username*s are specified, the entire queue is displayed; otherwise, only those jobs belonging to the named users are displayed. If the invoking user is not a privileged-user, then only jobs belonging to the invoking user are displayed.

The `-c` option sorts queue entires by their creation time, listing them consecutively by the time their `at` commands were given.

The `-n` option prints the total number of jobs currently in the queue, but does not list them.

FILES

`/var/spool/cron` spool area

SEE ALSO

at(AU_CMD), atrm(AU_CMD), cron(AU_CMD)

LEVEL

Level 1.

NAME

atrm – remove jobs spooled by at or batch

SYNOPSIS

`atrm` [`-f` | `-i`] [`-a`] *job-number* [[*job-number* | *username*] ...]

`atrm` [`-f` | `-i`] [`-a`] *username* [[*job-number* | *username*] ...]

DESCRIPTION

`atrm` removes delayed-execution jobs that were created with the `at` command. The list of jobs can be displayed by `atq` [see atq(AU_CMD)].

`atrm` removes each *job-number* you specify, and/or all jobs belonging to *username*, provided that the user owns the indicated jobs.

Jobs belonging to other users can only be removed by a user with appropriate privileges.

The `-f` option forces all information regarding the removal of the specified jobs to be suppressed.

The `-i` option is interactive; `atrm` asks if a job should be removed. A response of `y` verifies that the job is to be removed. Any other response cancels the removal.

The `-a` option removes jobs that were queued by the current user. If invoked by a user with appropriate privileges, the entire queue will be flushed.

FILES

`/var/spool/cron` spool area

SEE ALSO

at(AU_CMD), atq(AU_CMD), cron(AU_CMD).

LEVEL

Level 1.

NAME

chgrp – change the group ownership of a file

SYNOPSIS

chgrp [-R] [-h] *group file* **. . .**

DESCRIPTION

chgrp changes the group ID of the *files* given as arguments to *group*. The group may be either a decimal group ID or a group name found in the group ID file, **/etc/group**.

You must be the owner of the file, or have appropriate privilege to use this command.

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 a privileged user can arbitrarily change owner **ID**s whether this option is in effect or not.

chgrp has two options:

-R Recursive. **chgrp** descends through the directory, and any subdirectories, setting the specified group ID as it proceeds. When symbolic links are encountered, they are traversed.

-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.

The **LC_CTYPE** environment variable determines the codesets used in the arguments [see **LANG** on **envvar**(BA_ENV)].

FILES

/etc/group

/usr/lib/locale/*locale***/LC_MESSAGES/uxcore.abi**
language-specific message file [see **LANG** on **envvar**(BA_ENV)].

SEE ALSO

chmod(BA_OS), **chown**(BA_OS), **chown**(AU_CMD), **id**(AU_CMD), **group**(BA_ENV), **passwd**(AU_CMD)

LEVEL

Level 1.

NAME

chown – change file owner

SYNOPSIS

chown [-h] [-R] *owner[:group] file* **. . .**

DESCRIPTION

chown changes the owner of the *file*s to *owner*. The value of *owner* may be either a decimal user ID or a login name found in the **/etc/passwd** file. Login names in **/etc/passwd** must begin with a non-numeric character; an alphabetic character or any special character except colon is acceptable. **chown** will optionally also change the group ID of the *file*s to *group*. The value of *group* may be either a decimal group ID or a group name found in the group ID file **/etc/group**.

If **chown** is invoked by someone other than a privileged user, the set-user-ID bit of the file mode, 04000, is cleared.

Only the owner of a file (or a privileged user) may change the owner or group of that file.

Valid options to **chown** are:

-R Recursive. **chown** descends through the directory, and any subdirectories, setting the ownership (and group) ID as it proceeds. When symbolic links are encountered, they are traversed.

-h If the file is a symbolic link, change the owner (and group) of the symbolic link. Without this option, the owner (and group) of the file referenced by the symbolic link is changed. (See NOTICES below.)

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, and may change the group of the file to a group to which the owner belongs. Only a privileged user can arbitrarily change owner (and group) **ID**s whether this option is in effect or not. When this option is in effect, only a privileged user can arbitrarily change group **ID**s.

FILES

/etc/passwd

/usr/lib/locale/*locale***/LC_MESSAGES/uxcore.abi**

language-specific message file [see **LANG** on **envvar**(BA_ENV)].

SEE ALSO

chgrp(AU_CMD), **chmod**(BU_CMD), **chown**(BA_OS), **passwd**(AU_CMD)

LEVEL

Level 1.

NOTICES

chown(AU_CMD) does not check the user ID if it is in decimal form; to check a user ID in this form, you can use **chown**(BA_OS). In a Remote File Sharing environment, you may not have the permissions that the output of the **ls -l** command leads you to believe. For more information see the documentation on Network Administration.

Note that, with appropriate permissions, the owner of **`setuid`** files might inadvertantly be changed.

NAME

cron – clock daemon

SYNOPSIS

`cron`

DESCRIPTION

The command `cron`

NAME

crontab – user crontab file

SYNOPSIS

```
crontab [file]
crontab –e [username]
crontab –r [username]
crontab –l [username]
```

DESCRIPTION

The command `crontab` copies the specified file, or standard input if a file is not specified, into a directory that holds all users' crontabs. Note that if the Enhanced Security Extension is implemented, the file is stored at the Mandatory Access Control level of the user executing `crontab`.

The `–e` option edits a copy of the current user's `crontab` file, or creates an empty file to edit if the `crontab` file does not exist. When editing is complete, the file is installed as the user's `crontab` file. If *username* is given, the specified user's `crontab` file is edited, rather than the current user's `crontab` file; this may only be done by a user with the appropriate privileges.

The `-r` option removes a user's `crontab` from the `crontab` directory. Only a privileged user can specify a *username* to remove the `crontab` file of the specified user.

The option `-l` will list the `crontab` file of the invoking user. Only a privileged user can specify a *username* to list the `crontab` file of the specified user.

Users are permitted to use `crontab` if their names appear in the file `/etc/cron.d/cron.allow`. If that file does not exist, the file `/etc/cron.d/cron.deny` is checked to determine if the user should be denied access to `crontab`. If neither file exists, only a user with appropriate privileges is allowed to submit a job. If only `cron.deny` exists and is empty, global usage is permitted. The allow / deny files consist of one user name per line.

A `crontab` file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

minute (0–59),
hour (0–23),
day of the month (1–31),
month of the year (1–12),
day of the week (0–6 with 0=Sunday).

Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, each one is effective independent of the other. For example, `0 0 1,15 * 1` would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, `0 0 * * 1` would run a command only on Mondays).

The sixth field of a line in a `crontab` file is a string that is executed by the command interpreter at the specified times. A percent character in this field (unless escaped by \) is translated to a newline character. Only the first line (up to a % or end of line) of the command field is executed by the command interpreter. The other lines are made available to the command as standard input. The command `cron` supplies a default environment, defining the environment variables `HOME`, `LOGNAME`, and `PATH`.

Any line beginning with a # is a comment and will be ignored.

If you inadvertently enter the `crontab` command with no argument(s), do not attempt to get out with a CONTROL-D. This will cause all entries in your `crontab` file to be removed. Instead, interrupt the command.

If a privileged user modifies another user's `crontab` file, resulting behavior may be unpredictable. Instead, the privileged user should first `su` [see `su`(AU_CMD)] to the other user's login before making any changes to the `crontab` file.

FILES

`/etc/cron.d`	main cron directory
`/etc/cron.d/log`	accounting information
`/etc/cron.d/cron.allow`	list of allowed users
`/etc/cron.d/cron.deny`	list of denied users
`/var/spool/cron/crontabs`	spool area

USAGE

The new `crontab` file for a user overwrites an existing one.

Note: If standard output and standard error are not redirected, any generated output or errors will be mailed to the user.

SEE ALSO

sh(BU_CMD), su(AU_CMD), atq(AU_CMD), atrm(AU_CMD), cron(AU_CMD).

LEVEL

Level 1.

NAME

csplit – context split

SYNOPSIS

`csplit` [`-s`] [`-k`] [`-f`*prefix*] *file arg1* [*...argn*]

DESCRIPTION

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

`00`: From the start of *file* up to (but not including) the line referenced by *arg1*.

`01`: From the line referenced by *arg1* up to the line referenced by *arg2*.

.

.

.

n: From the line referenced by *argn* to the end of *file*.

If the *file* argument is a – then standard input is used.

The options to `csplit` are:

`-s` `csplit` normally prints the byte counts for each file created. If the `-s` option is used, `csplit` suppresses the printing of all byte counts.

`-k` `csplit` normally removes created files if an error occurs. If the `-k` option is used, `csplit` leaves previously created files intact.

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

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

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

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

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

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

All *rexp* type arguments that contain blanks or other characters meaningful to the shell should be enclosed in the appropriate quotes. Regular expressions may not contain embedded newlines. The command `csplit` does not affect the original file; it is the user's responsibility to remove it.

Note that the indicated size of the files created is in bytes, not the number of characters.

ERRORS

An error is reported if an argument does not reference a line between the current position and the end of the file.

USAGE

General.

EXAMPLE

This example creates four files, `cobol00` ... `cobol03`:

`csplit –fcobol` *file* `'/procedure division/' /par5./ /par16./`

After editing the split files, they can be recombined as follows:

`cat cobol0[0–3] >` *file*

Note that this example overwrites the original file.

The following example splits the file at every 100 lines, up to 10,000 lines:

```
csplit –k file  100  {99}
```

The `–k` option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

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

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

SEE ALSO

ed(BU_CMD), sh(BU_CMD).

LEVEL

Level 1.

NAME

cu – call another system

SYNOPSIS

cu [-s*speed*] [-l *line*] [-h] [-t] [-d][-o|-e] [-n] *telno*

cu [-s*speed*] [-h] [-d] [-o|-e] -l*line*

cu [-h] [-d] [-o|-e] *systemname*

DESCRIPTION

The command cu calls up another system, which will usually be a System V system, but may be a terminal, or a non-System V system. It manages an interactive conversation, with possible transfers of files.

The command cu accepts the following options and arguments:

-s*speed* Specifies the transmission speed. The default value is "Any" speed which will depend on the order of the lines in the system devices file.

-l*line* Specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line having the right speed. When the -l option is used without the -s option, the speed of a line is taken from the devices file. When the -l and -s options are both used together, cu will search the devices file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise, an error message will be printed and the call will not be made. If the specified device is associated with an auto dialer, a telephone number must be provided. Use of this option with *systemname* rather than *telno* is not allowed (see *systemname* below).

-h Emulates local echo. This option supports calls to other computer systems which expect terminals to be set to half-duplex mode.

-t Used to dial a terminal which has been set to auto answer. Appropriate mapping of carriage-return to carriage-return-linefeed pairs is set.

-d Causes diagnostic traces to be printed.

-o Designates that odd parity is to be generated for data sent to the remote system.

-e Designates that even parity is to be generated for data sent to the remote system.

-n For added security, this option prompts the user to provide the telephone number to be dialed rather than taking it from the command line.

telno When using an automatic dialer, *telno* is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately for delays of 4 seconds.

systemname

A *uucp* system name may be used rather than a telephone number; in this case, cu will obtain an appropriate direct line or telephone number from a system file. Note that the *systemname* option should not be used in conjunction with the -l and -s options as cu will connect to the first

available line for the system name specified, ignoring the requested line and speed.

After making the connection, `cu` runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output. Normally, an automatic `DC3/DC1` protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following user initiated commands:

`~.` terminate the conversation.

`~CTRL-Z` stop cu.

`~!` escape to an interactive command interpreter on the local system.

`~!`*cmd...* execute *cmd* on the local system

`~$`*cmd...* run *cmd* locally and send its output to the remote system for execution.

`~%cd` change the directory on the local system.

`~%take` *from* [*to*]
copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.

`~%put` *from* [*to*]
copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places.

`~~`*line* send the line ~*line* to the remote system.

`~%break` transmit a `BREAK` to the remote system (which can also be specified as `~%b`).

`~%debug` toggles the `-d` debugging option on or off (which can also be specified as `~%d`).

`~%nostop`
toggles between `DC3/DC1` input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the `DC3` and `DC1` characters.

The *receive* process normally copies data from the remote system to its standard output.

The use of `~%put` requires `stty` [see stty(AU_CMD)] and `cat` [see cat(BU_CMD)] on the remote side. It also requires that the current erase and kill characters on the remote system be identical to these current control characters on the local system. Backslashes are inserted at appropriate places.

The use of `~%take` requires the existence of `stty`, `cat`, and `echo` on the remote system. Also, `tabs` mode [see stty(AU_CMD)] should be set on the remote system if tabs are to be copied without expansion to spaces.

When cu is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using ~~. For example, uname can be executed on Z, X, and Y as follows (the response is given in brackets):

```
uname
[ Z ]
~[X]!uname
[ X ]
~~[Y]!uname
[ Y ]
```

In general, ~ causes the command to be executed on the original machine; ~~ causes the command to be executed on the next machine in the chain.

cu sets the input and output conversion mode to on or off, as appropriate, to avoid a character conversion on the local system when accessing the remote system.

On the remote system, the input and output conversion should be set manually, as cu cannot know whether input conversion is required or not. In most cases, remote systems can be used with input conversion on; however, when transferring files, it should be set to off before invoking the file transfer command in order to avoid unexpected conversion of the file contents.

ERRORS

Exit code is 0 for normal exit, otherwise, non-zero.

USAGE

End-user.

EXAMPLE

To dial a system whose telephone number is 9 1 201 555 2121 using 1200 baud (where dial tone is expected after the 9):

```
cu  -s 1200   9=12015552121
```

If the speed is not specified, "Any" is the default value.

To log in to a system connected by a direct line:

```
cu  -l  /dev/term/XX
```

or

```
cu -l term/XX
```

To dial a system with the specific line and a specific speed:

```
cu  -s 1200  -l  term/XX
```

To dial a system using a specific line associated with an auto dialer:

```
cu  -l  culXX  9=12015552121
```

To use a system name:

```
cu  systemname
```

SEE ALSO

cat(BU_CMD), echo(BU_CMD), stty(AU_CMD), uname(BU_CMD), uucp(AU_CMD).

LEVEL

Level 1.

NAME

dd – convert and copy a file

SYNOPSIS

dd [*option***=***value***] . . .**

DESCRIPTION

dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block sizes may be specified to take advantage of raw physical I/O. **dd** processes supplementary code set characters according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar**(BA_ENV)], except as noted below.

option	*values*
if=*file*	input file name; standard input is default.
of=*file*	output file name; standard output is default.
ibs=*n*	input block size *n* bytes (default 512).
obs=*n*	output block size *n* bytes (default 512).
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, preserve the input block size instead of packing short blocks into the output buffer (this is particularly efficient since no in-core copy need be done).
cbs=*n*	conversion buffer size (logical record length).
files=*n*	copy and concatenate *n* input files before terminating (makes sense only where input is a magnetic tape or similar device).
skip=*n*	skip *n* input blocks before starting copy (appropriate for magnetic tape, where *iseek* is undefined).
iseek=*n*	seek *n* blocks from beginning of input file before copying (appropriate for disk files, where *skip* can be slow).
oseek=*n*	seek *n* blocks from beginning of output file before copying.
seek=*n*	identical to *oseek*, retained for backward compatibility.
count=*n*	copy only *n* input blocks.
conv=ascii	convert EBCDIC to ASCII. Conversion results cannot be assured when supplementary code set characters are also subject to conversion.
	ebcdic convert ASCII to EBCDIC. Conversion results cannot be assured when supplementary code set characters are also subject to conversion.
	ibm slightly different map of ASCII to EBCDIC. Conversion results cannot be assured when supplementary code set characters are also subject to conversion.
conv=block	convert new-line terminated ASCII records to fixed length.

unblock	convert fixed length ASCII records to new-line terminated records.
lcase	map alphabetics to lower case. Multibyte characters are not converted.
ucase	map alphabetics to upper case. Multibyte characters are not converted.
swab	swap every pair of bytes.
noerror	do not stop processing on an error (limit of 5 consecutive errors).
sync	pad every input block to *ibs*.
... , ...	several comma-separated conversions.

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by **x** to indicate multiplication.

cbs is used only if **ascii**, **unblock**, **ebcdic ibm**, or **lock** conversion is specified. In the first two cases, *cbs* characters are copied into the conversion buffer, any specified character mapping is done, trailing blanks are trimmed, and a new-line is added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer and blanks are added to make up an output record of size *cbs*. If **cbs** is unspecified or zero, the **ascii**, **ebcdic**, and **ibm** options convert the character set without changing the block structure of the input file; the **unblock** and **block** options become a simple file copy.

After completion, **dd** reports the number of whole and partial input and output blocks.

USAGE

This command will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per tape block into the ASCII file *x*:

```
dd  if=/dev/rmt*  of=x  ibs=800  obs=8k  cbs=80  conv=ascii,lcase
```

Note the use of raw magnetic tape. **dd** is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary block sizes. Note also that **/rmt*** represents the raw magnetic tape device name.

Reading from magnetic tape in any fixed-length block length besides the block length that the media was written in originally will cause an I/O error. If you want to read a tape that was written using a block-length besides the default of 512, you must use the **tapecntl**(1) command (qv) to either set the block-length of the drive to match the block length of the media or to set the drive into variable block length mode.

Errors

f+*p* **records in(out)**	numbers of full and partial blocks read(written)

Files

/usr/lib/locale/*locale***/LC_MESSAGES/uxcore.abi**
language-specific message file [see **LANG** on **envvar**(BA_ENV)].

SEE ALSO

cp(BU_CMD)

LEVEL

Level 1.

NOTICES

Reading from magnetic tape in any fixed-length block length, besides the block length that the media was written in originally, will cause an I/O error. In order to read a tape that was written using some block length besides the default of 512, use the **tapecntl**(1) command (**qv**) to either set the block length of the drive to match the block length of the media, or to set the drive into variable block length mode.

Do not use **dd** to copy files between file systems having different block sizes.

dd does not always require block sizes that are in multiples of 512 bytes. Block size is device dependent. If input data blocks are not a multiple of 512, however, the read side will have no error messages, but the write side might have a ''write error'' message. **dd** transfers correctly if the input data block sizes are a multiple of 512.

Using a blocked device to copy a file will result in extra nulls being added to the file to pad the final block to the block boundary.

Using **dd** with a cartridge tape is not recommended.

Using variable-length block mode when writing magnetic tapes is discouraged because it may not work correctly in releases before SVR4.2 MP. Magnetic tape should always be written in fixed-length block mode, even though you are free to change the default fixed-block length from 512 bytes to any other fixed-block mode the tape drive supports.

NAME

dircmp – directory comparison

SYNOPSIS

dircmp [-d] [-s] [-w*n***]** *dir1 dir2*

DESCRIPTION

dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the file names common to both directories have the same contents. **dircmp** processes supplementary code set characters in directory and file names according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar**(BA_ENV)].

-d Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in **diff**(BU_CMD).

-s Suppress messages about identical files.

-w*n* Change the width of the output line to *n* columns. The default width is 72.

FILES

/usr/lib/locale/*locale***/LC_MESSAGES/uxdfm**
language-specific message file [see **LANG** on **envvar**(BA_ENV)].

SEE ALSO

cmp(BU_CMD), **diff**(BU_CMD), **pr**(BU_CMD)

LEVEL

Level 1.

NAME

ex – text editor

SYNOPSIS

`ex [-] [-v] [-r] [-R] [+`*command*`] [-`*l*`] [`*file* ...`]`

DESCRIPTION

The command `ex` is a line oriented text editor, which supports both command and display editing [see vi(AU_CMD)]. The command line options are:

`-` Suppresses all interactive-user feedback. This is useful in processing editor scripts.

`-v` Invokes `vi`

`-r` Recovers the named files after an editor or system crash. If no files are named, a list of all saved files is printed.

`-R` *Read-only* mode set, prevents accidentally overwriting the file.

`+`*command*
Begins editing by executing the specified editor search or positioning *command*.

`-l` `LISP` mode; indents appropriately for *lisp* code; the `()` `{}` `[[` and `]]` commands in `vi` are modified to have meaning for lisp.

The *file* argument(s) indicates files to be edited in the order specified.

The name of the file being edited by `ex` is the *current* file. The text of the file is read into a buffer, and all editing changes are performed in this buffer; changes have no effect on the file until the buffer is written out explicitly.

The *alternate* filename is the name of the last file mentioned in an editor command, or the previous current filename if the last file mentioned became the current file. The character % in filenames is replaced by the current filename, and the character #, by the alternate filename.

The named buffers, ASCII `a` through `z`, may be used for saving blocks of text during the edit. If the buffer name is specified in upper case, the buffer is appended to rather than being overwritten.

The read-only mode can be cleared from within the edit by setting the `noreadonly` edit option (see `Edit Options` below). Writing to a different file is allowed in read-only mode; in addition, the write can be forced by using `!` (see the `write` command below).

When an error occurs, `ex` sends the `BEL` character to the terminal (to sound the bell) and prints a message. If an interrupt signal is received, `ex` returns to the command level, in addition to the above actions. If the editor input is from a file, `ex` exits at the interrupt. (The bell action may be disabled by the use of an edit option; see below.)

If the system crashes, `ex` attempts to preserve the buffer if any unwritten changes were made. The command line option `-r` is used to retrieve the saved changes.

At the beginning, `ex` is in the *command* mode, which is indicated by the `:` prompt. The `input` mode is entered by `append`, `insert`, or `change` commands; it is left (and command mode re-entered) by typing a period (`.`) alone at the beginning of a line.

Command lines beginning with the double quote character (") are ignored. (This may be used for comments in an editor script.)

Addressing

`.` Dot (`.`) refers to the current line. There is always a current line; the positioning may be the result of an explicit movement by the user, or the result of a command that affected multiple lines (in which case it is usually the last line affected).

n The *n*th line in the buffer, with lines numbered sequentially from 1.

`$` The last line in the buffer.

`%` Abbreviation for `1,$`, the entire buffer.

+*n*
–*n* An offset relative to the current line. (The forms `.+3`, `+3`, and `+++` are equivalent.)

/*pat*/
?*pat*? Line containing the pattern (regular expression) *pat*, scanning forward (`//`) or backward (`??`). The trailing `/` or `?` may be omitted if the line is only to be printed. If the pattern is omitted, the previous pattern specified is used.

`'`*x* Lines may be marked using single lower case ASCII letters (see the `mark` command below); `'`*x* refers to line marked *x*. In addition, the previous current line is marked before each non-relative motion; this line may be referred to by using `'` for *x*.

Addresses to commands consist of a series of line addresses (specified as above), separated by a comma (`,`) or a semicolon (`;`). Such address lists are evaluated left-to-right. When a semicolon (`;`) is the separator, the current line is set to the value of the previous address before the next address is interpreted. If more addresses are given than the command requires, then all but the last one or two are ignored. Where a command requires two addresses, the first line must precede the second one in the buffer. A null address in a list defaults to the current line.

Command names and abbreviations

abbrev	`ab`	next	`n`	unmap	`unm`
append	`a`	number	`# nu`	version	`ve`
args	`ar`	preserve	`pre`	visual	`vi`
change	`c`	print	`p`	write	`w`
copy	`co`	put	`pu`	xit	`x`
delete	`d`	quit	`q`	yank	`ya`
edit	`e`	read	`re`	(window)	`z`
file	`f`	recover	`re`	(escape)	`!`
global	`g v`	rewind	`rew`	(lshift)	`<`

insert	`i`	set	`se`	(rshift)	`>`
join	`j`	shell	`sh`	(resubst)	`& s`
list	`l`	source	`so`	(scroll)	`^D`
map	`map`	substitute	`s`	(line no)	`=`
mark	`k` *or* `ma`	unabbrev	`una`		
move	`m`	undo	`u`		

Command descriptions

In the following, *line* is a single line address, given in any of the forms described in the **Addressing** section above; *range* is a pair of line addresses, separated by a comma or semicolon (see the **Addressing** section for the difference between the two); *count* is a positive integer, specifying the number of lines to be affected by the command; *flags* is one or more of the characters `#`, `p`, and `l`; the corresponding command to print the line is executed after the command completes. Any number of `+` or `-` characters may also be given with these flags.

When *count* is used, *range* is not effective; only a line number should be specified instead, to indicate the first line affected by the command. (If a range is given, then the last line of the range is taken as the starting line for the command.)

These modifiers are all optional; the defaults are as follows, unless otherwise stated: the default for *line* is the current line; the default for *range* is the current line only (`.,.`); the default for *count* is `1`; the default for *flags* is null.

When only a *line* or a *range* is specified (with a null command), the implied command is `print`; if a null line is entered, the next line is printed (equivalent to `.+1p`)

`ab` *word rhs*
: Adds the named abbreviation to the current list. In visual mode, if *word* is typed as a complete word during input, it is replaced by the string *rhs*.

line `a` Enters input mode; places the input text after the specified line. If line 0 is specified, the text is placed at the beginning of the buffer. The last input line becomes the current line, or the target line, if no lines are input.

`ar` Prints the argument list with the current argument inside `[` and `]`.

range `c` *count*
: Enters input mode; the input text replaces the specified lines. The last input line becomes the current line; if no lines are input, the current line becomes the line before the target line, or the first line of the file if there are no lines preceding the target.

range `co` *line flags*
: Places a copy of the specified lines (*range*) after the specified destination *line*; line 0 specifies that the lines are to be placed at the beginning of the buffer.

range `d` *buffer count*
: Deletes the specified lines from *buffer*. If a named buffer is specified, the deleted text is saved in it. The line after the deleted lines becomes the current line, or the last line if the deleted lines were at the end.

`e` *+line file*
Begins editing a new file. If the current buffer has been modified since the last write, then a warning is printed and the command is aborted. This action may be overridden by appending the character `!` to the command (*e.g.*, `e!` *file*). The current line is the last line of the buffer; however, if this command is executed from within `visual`, the current line is the first line of the buffer. If the *+line* option is specified, the current line is set to the specified position, where *line* may be a number (or `$`) or specified as */pat* or *?pat*.

`f` Prints the current filename and other information, including the number of lines and the current position.

range `g` */pat/ cmds*
First marks the lines within the given range that match the given pattern. Then the given command(s) is executed with `.` set to each marked line.

cmds may be specified on multiple lines by hiding newlines with a backslash. If *cmds* are omitted, each line is printed. For an `append`, `change`, or `insert` command, the terminating dot may be omitted if it ends *cmds*. `visual` commands are also permitted, and take input from the terminal.

The `global` command itself, and the `undo` command are not allowed in *cmds*. The edit options `autoprint`, `autoindent` and `report` are inhibited.

range `v` */pat/ cmds*
This is the same as the `global` command, except that `cmds` is run on the lines that *do not* match the pattern.

line `i` Enters input mode; the input text is placed before the specified line. The last line input becomes the current line, or the line before the target line, if no lines were input.

range `j` *count flags*
Joins the text from the specified lines together into one line. White space is adjusted to provide at least one blank character, two if there was a period at the end of the line, or none if the first following character is a right parentheses [`)`]. Extra white space at the start of a line is discarded.

Appending the command with a `!` causes a simpler join with no white space processing.

range `l` *count flags*
Prints the specified lines with tabs printed as `^I` and the end of each line marked with a trailing `$`. (The only useful flag is #, for line numbers.) The last line printed becomes the current line.

`map` *x rhs*
The `map` command defines macros for use in `visual` mode. The first argument is a single character, or the sequence *#n*, where *n* is a digit, to refer to the function key *n*. When this character or function key is typed in `visual` mode, the action is as if the corresponding *rhs* had been typed. If `!` is appended to the command `map`, then the mapping is effective during insert

mode rather than command mode. Special characters, white space, and newline must be escaped with a control-V to be entered in the arguments.

line `ma` *x*
(The letter `k` is an alternative abbreviation for the `mark` command.) The specified line is given the specified mark *x*, which must be a single ASCII lower case letter. (The *x* must be preceded by a space or tab.) The current line position is not affected.

range `m` *line*
Moves the specified lines (*range*) after the target line. The first of the moved lines becomes the current line.

`n` Edits the next file from the command line argument list. Appending a `!` to the command overrides the warning about the buffer having been modifed since the last write (discarding any changes). The argument list may be replaced by specifying a new one on this command line.

range `nu` *count flags*
(The character # is an alternative abbreviation for the `number` command.) Prints the lines, each preceded by its line number. (The only useful flag is `l`.) The last line printed becomes the current line.

`pre` The current editor buffer is saved as though the system had just crashed. This command is for use in emergencies, for example when a write does not work, and the buffer cannot be saved in any other way.

range `p` *count*
Prints the specified lines, with non-printing characters printed as control characters in the form `^x`; `DEL` is represented as `^?`. The last line printed becomes the current line.

line `pu` *buffer*
Puts back deleted or "yanked" lines. A buffer may be specified; otherwise, the text in the unnamed buffer (where deleted or yanked text is placed by default) is restored.

`q` Causes termination of the edit. If the buffer has been modified since the last write, a warning is printed and the command fails. This warning may be overridden by appending a `!` to the command (discarding changes).

line `r` *file*
Places a copy of the specified file in the buffer after the target line (which may be line 0 to place text at the beginning). If no *file* is named the current file is the default. If there is no current file then *file* becomes the current file. The last line read becomes the current line; in `visual` the first line read becomes the current line.

If *file* is given as `!`*string* then *string* is taken to be a system command, and passed to the command interpreter; the resultant output is read into the buffer. A blank or tab must precede the `!`.

`rec` *file*
Recovers *file* from the save area, after an accidental hangup or a system crash.

`rew` Rewinds the argument list, and edits the first file in the list. Warnings may be overridden by appending a `!`.

`se` *parameter*
With no arguments, the set command prints those options whose values have been changed from the default settings; with the parameter `all`, it prints all of the option values.

Giving an option name followed by a `?` causes the current value of that option to be printed. The `?` is necessary only for Boolean valued options. Boolean options are given values by the form `se` *option* to turn them on, or `se no`*option* to turn them off; string and numeric options are assigned by the form `se` *option*=*value*. More than one parameter may be given; they are interpreted left to right.

See **Edit Options** below for further details about options.

`sh` Puts the user into the command interpreter [usually `sh`; see sh(BU_CMD)]; editing is resumed on exit.

`so` *file*
Reads and executes commands from the specified file. `so` commands may be nested.

range `s` */pat/repl/ options count flags*
On each specified line, the first instance of the pattern *pat* is replaced by the string *repl*. (See **Regular Expressions** and **Replacement Strings** below.) If *options* includes the letter `g` (global), then all instances of the pattern in the line are substituted. If the option letter `c` (confirm) is included, then before each substitution the line is typed with the pattern to be replaced marked with ^ characters; a response of `y` causes the substitution to be done, while any other input aborts it. The last line substituted becomes the current line.

`una` *word*
Deletes *word* from the list of abbreviations.

`u` Reverses the changes made by the previous editing command. For this purpose, `global` and `visual` are considered single commands. Commands which affect the external environment, such as `write`, `edit` and `next`, cannot be undone. An `undo` can itself be reversed.

`unm` *x*
Removes the macro definition for *x*.

`ve` Prints the current version of the editor.

line `vi` *type count*
Enters visual mode at the specified line. The *type* is optional, and may be `-` or `.`, as in the `z` command, to specify the position of the specified line on the screen window. (The default places the line at the top of the screen window.) A *count* specifies an initial window size; the default is the value of the edit option `window`. The command `Q` exits visual mode. [For more information, see vi(AU_CMD)]

range `w` *file*
Writes the specified lines (the whole buffer, if *range* is not given) out to *file*, printing the number of lines and characters written. If *file* is not specified, the default is the current file. (The command fails with an error message if there is no current file and no file is specified.)

If an alternate file is specified, and the file exists, then the write will fail; it may be forced by appending a `!` to the command. An existing file may be appended to by appending `>>` to the command. If the file does not exist, an error is reported.

If the file is specified as `!string`, then *string* is taken as a system command; the command interpreter is invoked, and the specified lines are passed as standard input to the command.

The command `wq` is equivalent to a `w` followed by a `q`; `wq!` is equivalent to `w!` followed by `q`.

`x` Writes out the buffer if any changes have been made, and then (in any case) quits.

range `ya` *buffer count*
Places the specified lines in the named buffer. If *buffer* is not specified, the unnamed buffer is used (where the most recently deleted or yanked text is placed by default).

line `z` *type count*
If *type* is omitted, then *count* lines following the specified line (default current line) are printed. The default for *count* is the value of the edit option `window` if you are in visual mode. If you are not in visual mode, the default count is the window length (number of lines in the window).

If *type* is specified, it must be `-` or `.`; a `-` causes the line to be placed at the bottom of the screen, while a `.` causes the line to be placed in the middle. The last line printed becomes the current line.

`!` *command*
The remainder of the line after the `!` is passed to the system command interpreter for execution. A warning is issued if the buffer has been changed since the last write. A single `!` is printed when the command completes. The current line position is not affected.

Within the text of *command*, `%` and `#` are expanded as filenames, and `!!` is replaced with the text of the previous `!` command. (Thus `!!` repeats the previous `!` command.) If any such expansion is done, the expanded line is echoed.

range `!` *command*
In this form of the `!` command, the specified lines (there is no default; see previous paragraph) are passed to the command interpreter as standard input; the resulting output replaces the specified lines.

range `<` *count*
Shifts the specified lines to the left; the number of spaces to be shifted is determined by the edit option `shiftwidth`. Only white space (blanks and tabs) is lost in shifting; other characters are not affected. The last line

changed becomes the current line.

range `>` *count*
Shifts the specified lines to the right, by inserting white space (see previous paragraph for further details).

range `&` *options count flags*
Repeats the previous substitute command, as if & were replaced by the previous `s/pat/repl/`. (The same effect is obtained by omitting the `/pat/repl/` string in the `substitute` command.)

`CTRL-D`
`CTRL-D` (ASCII `EOT`) prints the next *n* lines, where *n* is the value of the edit option `scroll`.

line `=` Prints the line number of the specified line (default last line). The current line position is not affected.

Regular Expressions

Regular expressions are interpreted according to the setting of the edit option `magic`; the following assumes the setting `magic`. The differences caused by setting `nomagic` are described below.

`vi` [see vi(AU_CMD)] regular expressions are the same as `ed` [see ed(BU_CMD)] except for the following differences:

`[`*string*`]`
Matches any single character in *string*. Within *string*, the following have special meanings: a pair of characters separated by `-` defines a range (*e.g.*, `[a-z]` defines any ASCII lower case letter); the character `^`, if it is the first one in *string*, causes the construct to match characters other than those specified in *string*. These special meanings can be removed by escaping the characters with \.

`~` Matches the replacement part of the last `substitute` command. The special meaning can be removed by escaping with \.

A concatenation of two regular expressions is a regular expression that matches the concatenation of the strings matched by each component.

When `nomagic` is set, the only characters with special meanings are `^` at the beginning of a pattern, `$` at the end of a pattern, and \. The characters `.`, `*`, `[`, and `~` lose their special meanings, unless escaped by a \.

Replacement Strings

The character & (`\&` if `nomagic` is set) in the replacement string stands for the text matched by the pattern to be replaced. The character `~` (`\~` if `nomagic` is set) is replaced by the replacement part of the previous `substitute` command. The sequence `\`*n* where *n* is an integer, is replaced by the text matched by the pattern enclosed in the *n*th set of parentheses `\(` and `\)`. The sequence `\u` (`\l`) causes the immediately following character in the replacement to be converted to upper case (lower case), if this character is a letter. The sequence `\U` (`\L`) turns such conversion on, until the sequence `\E` or `\e` is encountered, or the end of the replacement string is reached.

Edit Options

The command ex has several options that modify its behavior. These options have default settings, which may be changed using the set command (see above). Options may also be set at startup by putting a set command string in the environment variable EXINIT, or in the file .exrc in the HOME directory.

Options are Boolean unless otherwise specified.

autoindent, ai
: If autoindent is set, each line in insert mode is indented (using blanks and tabs) to align with the previous line. (Starting indentation is determined by the line appended after, or the line inserted before, or the first line changed.) Additional indentation can be provided as usual; succeeding lines will automatically be indented to the new alignment. Reducing the indent is achieved by typing CTRL-D one or more times; the cursor is moved back shiftwidth spaces for each CTRL-D. (A ^ followed by a CTRL-D removes all indentation temporarily for the current line; a 0 followed by a CTRL-D removes all indentation.)

autoprint, ap
: The current line is printed after each command that changes buffer text. (autoprint is suppressed in globals.)

autowrite, aw
: The buffer is written (to the current file) if it has been modified, and a next, rewind, or ! command is given.

beautify, bf
: Causes all control characters other than tab, newline, and formfeed to be discarded from the input text.

directory, dir
: The value of this option specifies the directory in which the editor buffer is to be placed. If this directory can not be written to by the user, the editor quits.

edcompatible, ed
: Causes the presence of g and c suffixes on substitute commands to be remembered, and toggled by repeating the suffixes. For example, "1s/a/A/g" followed by "2s" will substitue all instances of "a" for "A" on line 2.

ignorecase, ic
: All upper case characters in the text are mapped to lower case in regular expression matching. Also, all upper case characters in regular expressions are mapped to lower case.

lisp
: autoindent mode, and the () { } [[ ]] commands in visual are suitably modified for lisp code.

list
: All printed lines are displayed with tabs shown as ^I, and the end of line marked by a $.

`magic`
Changes interpretation of characters in regular expressions and substitution replacement strings (see the relevant sections above).

`number, nu`
Causes lines to be printed with line numbers.

`paragraphs, para`
The value of this option is a string, in which successive pairs of characters specify the names of text-processing macros which begin paragraphs. (A macro appears in the text in the form `.`*XX*, where the `.` is the first character in the line.)

`prompt`
When set, command mode input is prompted for with a colon (`:`); when unset, no prompt is displayed.

`redraw`
The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a large amount of output to the terminal, it is useful only at high transmission speeds.)

`remap`
If set, then macro translation allows for macros defined in terms of other macros; translation continues until the final product is obtained. If unset, then a one-step translation only is done.

`report`
The value of this option gives the number of lines that must be changed by a command before a report is generated on the number of lines affected.

`scroll`
The value of this option determines the number of lines scrolled on a `CTRL-D`, and the number of lines displayed by the `z` command (twice the value of scroll).

`sections`
The value of this option is a string, in which successive pairs of characters specify the names of text-processing macros which begin sections. (See `paragraphs` option above.)

`shiftwidth, sw`
The value of this option gives the width of a software tab stop, used during `autoindent`, and by the shift commands.

`showmatch, sm`
In `visual` mode, when a `)` or `}` is typed, the matching `(` or `{` is shown if it is still on the screen.

`slowopen, slow`
In `visual` mode, prevents screen updates during input to improve throughput on unintelligent terminals.

`tabstop, ts`
The value of this option specifies the software tab stops to be used by the editor to expand tabs in the input file.

`terse`
When set, error messages are shorter.

`window`
The number of lines in a text window in `visual` mode.

`wrapscan, ws`
When set, searches (using `//` or `??`) wrap around the end of the file; when unset, searches stop at the beginning or the end of the file, as appropriate.

`wrapmargin, wm`
In `visual` mode, if the value of this option is greater than zero (say *n*), then a newline is automatically added to an input line, at a word boundary, so that lines end at least *n* spaces from the right margin of the terminal screen.

`writeany, wa`
Inhibits the checks otherwise made before write commands, allowing a write to any file (provided the system allows it).

FILES

`$HOME/.exrc`	editor initialization file

USAGE

End-user.

The `undo` command causes all marks to be lost on lines that were changed and then restored.

The `z` command prints a number of logical rather than physical lines. More than a screen-ful of output may result if long lines are present.

Null characters are discarded in input files and cannot appear in resultant files.

SEE ALSO

ed(BU_CMD), vi(AU_CMD), terminfo(TI_ENV).

FUTURE DIRECTIONS

To conform to the command syntax standard, the `+`*command* option will be changed to the form `-c`*command*. The old form will continue to be accepted for some time.

LEVEL

Level 1.

The option `+`*command* is Level 2, effective September 30, 1989.

NAME

gencat – generate a formatted message catalogue

SYNOPSIS

gencat [**-m**] *catfile msgfile* ...

DESCRIPTION

The **gencat** utility merges the message text source file(s) *msgfile* into a formatted message database *catfile*. The database *catfile* will be created if it does not already exist. If *catfile* does exist its messages will be included in the new *catfile*. If set and message numbers collide, the new message text defined in *msgfile* will replace the old message text currently contained in *catfile*. The message text source file (or set of files) input to **gencat** can contain either set and message numbers or simply message numbers, in which case the set **NL_SETD** [see **nl_types**(BA_ENV)] is assumed.

The format of a message text source file is defined as follows. Note that the fields of a message text source line are separated by a single ASCII space or tab character. Any other ASCII spaces or tabs are considered as being part of the subsequent field.

$set *n comment*

Where *n* specifies the set identifier of the following messages until the next **$set**, **$delset** or end-of-file appears. *n* must be a number in the range (1–{**NL_SETMAX**}). Set identifiers within a single source file need not be contiguous. Any string following the set identifier is treated as a comment. If no **$set** directive is specified in a message text source file, all messages will be located in the default message set **NL_SETD**.

$delset *n comment*

Deletes message set *n* from an existing message catalogue. Any string following the set number is treated as a comment.

(Note: if *n* is not a valid set it is ignored.)

$ *comment*

A line beginning with a dollar symbol (**$**) followed by an ASCII space or tab character is treated as a comment.

m message text

The *m* denotes the message identifier, which is a number in the range (1-{**NL_MSGMAX**}). The message text is stored in the message catalogue with the set identifier specified by the last **$set** directive, and with message identifier *m*. If the message text is empty, and an ASCII space or tab field separator is present, an empty string is stored in the message catalogue. If a message source line has a message number, but neither a field separator nor message text, the existing message with that number (if any) is deleted from the catalogue. Message identifiers need not be contiguous. The length of message text must be in the range (0–{**NL_TEXTMAX**}).

$quote *c*

This line specifies an optional quote character *c*, which can be used to surround message text so that trailing spaces or null (empty) messages are visible in a message source line. By default, or if an empty **$quote** directive is supplied, no quoting of message text will be recognized.

Empty lines in a message text source file are ignored.

Text strings can contain the special characters and escape sequences defined in the following table:

Description	Symbol	Sequence
newline	NL(LF)	\n
horizontal tab	HT	\t
vertical tab	VT	\v
backspace	BS	\b
carriage return	CR	\r
form feed	FF	\f
backslash	\	\\
bit pattern	ddd	\ddd

If the character following a backslash is not one of those specified, the backslash is ignored. The escape sequence **\ddd** consists of backslash followed by 1, 2, or 3 octal digits, which are taken to specify the value of the desired character.

Backslash followed by an ASCII newline character is also used to continue a string on the following line. Thus, the following two lines describe a single message string:

```
1 This line continues \
to the next line
```

which is equivalent to:

```
1 This line continues to the next line
```

SEE ALSO

mkmsgs(AS_CMD) **catopen**(BA_LIB), **catgets**(BA_LIB), **catclose**(BA_LIB), **gettxt**(BA_LIB), **nl_types**(BA_ENV).

LEVEL

Level 1.

NAME

groups – show group memberships

SYNOPSIS

`groups` [*user*]

DESCRIPTION

The `groups` command shows the groups to which you or the optionally specified *user* belong. Each *user* belongs to a group specified in the password file `/etc/passwd` and possibly to other groups as specified in the file `/etc/group`. If you do not own a file but belong to the group by which it is owned, then you are granted group access to the file.

FILES

`/etc/passwd`
`/etc/group`

SEE ALSO

getgroups(BA_OS).

LEVEL

Level 1.

NAME

iconv – code set conversion utility

SYNOPSIS

iconv -f *fromcode* **-t** *tocode* **[***file***]**

DESCRIPTION

iconv converts the characters or sequences of characters in *file* from one code set to another and writes the results to standard output. Should no conversion exist for a particular character then it is converted to the underscore '_' in the target code set.

Your system must have the appropriate data files for **iconv** to work. The European Language Supplement (ELS) provides a large selection of these data files.

The required arguments *fromcode* and *tocode* identify the input and output code sets, respectively. If no *file* argument is specified on the command line, **iconv** reads the standard input.

iconv will always convert to or from the ISO 8859-1 Latin alphabet No.1, from or to an ISO 646 ASCII variant code set for a particular language. The ISO 8859-1 code set will support the majority of 8-bit code sets. The conversions attempted by **iconv** accommodate the most commonly used languages.

The following table lists the supported conversions.

Code Set Conversions Supported				
Code	Symbol	Target Code	Symbol	comment
ISO 646	646	ISO 8859-1	8859	US ASCII
ISO 646de	646de	ISO 8859-1	8859	German
ISO 646da	646da	ISO 8859-1	8859	Danish
ISO 646en	646en	ISO 8859-1	8859	English ASCII
ISO 646es	646es	ISO 8859-1	8859	Spanish
ISO 646fr	646fr	ISO 8859-1	8859	French
ISO 646it	646it	ISO 8859-1	8859	Italian
ISO 646sv	646sv	ISO 8859-1	8859	Swedish
ISO 8859-1	8859	ISO 646	646	7 bit ASCII
ISO 8859-1	8859	ISO 646de	646de	German
ISO 8859-1	8859	ISO 646da	646da	Danish
ISO 8859-1	8859	ISO 646en	646en	English ASCII
ISO 8859-1	8859	ISO 646es	646es	Spanish
ISO 8859-1	8859	ISO 646fr	646fr	French
ISO 8859-1	8859	ISO 646it	646it	Italian
ISO 8859-1	8859	ISO 646sv	646sv	Swedish

EXAMPLES

Assuming the ELS is installed on your system, the following converts the contents of file **mail1** from code set 8859 to 646fr and stores the results in file **mail.local**.

```
iconv -f 8859 -t 646fr mail1 > mail.local
```

FILES

/usr/lib/iconv/iconv_data
lists the conversions supported

/usr/lib/iconv/*
conversion tables, if any

/usr/lib/locale/*locale***/LC_MESSAGES/uxmesg**
language-specific message file [See **LANG** on **envvar**(BA_ENV).]

Errors

iconv returns 0 upon successful completion, non-zero otherwise.

USAGE

Administrator.

SEE ALSO

iconv(BA_LIB), **iconv_close**(BA_LIB), **iconv_open**(BA_LIB)

LEVEL

Level 1.

NAME

id – print the user name and ID, and group name and ID

SYNOPSIS

id [*user***]**

id -G [-n] [*user***]**

id -g [-nr] [*user***]**

id -u [-nr] [*user***]**

id [-a]

DESCRIPTION

id displays information on the calling process's user and group **ID**s. If *user* is specified, then information on the *user* login is displayed instead.

The information displayed is the user **ID** and name and the group **ID** and name. If the real and effective user **ID**s do not match, both are printed. The same is true for real and effective group **ID**s. If the user belongs to more than one group, the extra groups are also displayed.

The output format is a sequence of equality statements of the form, ''*id-title***=***numeric-id***(***name***)**'', each separated by a singlespace. The *numeric-id* is the numeric representation of the group or user, real or effective **ID**. The *name* is the symbolic representation of this **ID** (the login name of the user, for example). In the C locale, the *id-title* is one of the following strings.

id-title	description
uid	user **ID**
gid	group **ID**
euid	effective user **ID**
egid	effective group **ID**
groups	supplementary group **ID**s

The equality statements are presented in the order given in the table, with each statement present according to the conditions explained in the first paragraph. However, the **groups** statement, which is always written last, if at all, is slightly different because each extra supplementary group after the first does not have its own equality statement but adds ''**,** *numeric-id* **(***name***)**'' to the end of the output line.

Options

-a This option does nothing. It is for backward compatibility only, and should not be used.

-G Display all different (numeric) group **ID**s (real, effective, and supplementary), each separated by a **space**.

-g Display only the effective (numeric) group **ID**.

-n Display the name instead of the numeric **ID** (when used with **-G**, **-g**, or **-u**).

-r Display the real **ID** instead of the effective **ID** (when used with **-g** or **-u**).

-u Display only the effective (numeric) user **ID**.

SEE ALSO

logname(AU_CMD), getuid(BA_OS)

FUTURE DIRECTIONS

The **-a** will be removed in the next issue of the SVID. This flag is no longer necessary. Notice must be taken of the default behavior of the command which has changed for POSIX 1003.2 conformance, and may affect some scripts that parse the output of the **id** command on systems that support multiple group affiliations.

LEVEL

Level 1. The **-a** flag is moved to Level 2, effective September 30, 1993.

NAME

join – relational database operator

SYNOPSIS

join [-a *file_no* **| -v** *file_no***] [-e** *string***] [-o** *list***] [-t** *char***]**
[-1 *field***] [-2** *field***]** *file1 file2*

DESCRIPTION

join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If either *file1* or *file2* is **-**, the standard input is used. *file1* and *file2* must be sorted in increasing code set collating sequence as specified by the **LC_COLLATE** environmental variable on the fields on which they are to be joined, normally the first in each line [see **sort**(BU_CMD)]. **join** processes supplementary code set characters in files, and recognizes supplementary code set characters given to the **-e** and **-t** options according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar**(BA_ENV)].

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the options below use the argument *file_no*. This argument should be a **1** or a **2** referring to either *file1* or *file2*, respectively.

OPTIONS

-a *file_no*

-1 *field* Join on the *field*th field of *file1*. Fields are positive decimal integers starting with **1**.

-2 *field* Join on the *field*th field of *file2*. Fields are positive decimal integers starting with **1**.

EXAMPLES

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name, and the login directory. It is assumed that the files have been sorted in code set collating sequence on the group ID fields.

```
join -1 4 -2 3 -o 1.1 2.1 1.6 -t : /etc/passwd /etc/group
```

FILES

/usr/lib/locale/*locale***/LC_MESSAGES/uxdfm**
language-specific message file [see **LANG** on **envvar**(BA_ENV)].

SEE ALSO

awk (BU_CMD), **comm** (BU_CMD), **sort** (BU_CMD), **uniq** (BU_CMD)

LEVEL

Level 1.

NOTICES

With default field separation, the collating sequence is that of **sort -b**; with **-t**, the sequence is that of a plain sort.

The conventions of the **join**, **sort**, **comm**, **uniq**, and **awk** commands are wildly incongruous.

Filenames that are numeric may cause conflict when the **-o** option is used just before listing filenames.

The **-j, -j1,** and **-j2** options have been made obsolete by POSIX. It is recommended that application authors avoid using these options.

NAME

logname – get login name

SYNOPSIS

logname

DESCRIPTION

logname returns the name of the user running the process.

The **LC_CTYPE** environment variable defines the codesets that are used in the user name. [See **LANG** in **envvar** (BA_ENV).]

FILES

/etc/profile

/usr/lib/locale/*locale***/LC_MESSAGES/uxue.abi**

language-specific message file [See **LANG** in **envvar** (BA_ENV)].

SEE ALSO

env (SD_CMD), **envvar** (BA_ENV)

LEVEL

Level 1.

NAME

lp, **cancel** – send / cancel print requests

SYNOPSIS

lp [*print-options***] [***files***]**

lp -i *request-ID print-options*

cancel [*request-IDs***] [***printers***]**

cancel -u *login-IDs* **[***printers***]**

DESCRIPTION

The first form of the **lp** command arranges for the named *files* and associated information (collectively called a request) to be printed. If filenames are not specified on the command line, the standard input is assumed. The standard input may be specified along with named *files* on the command line by listing the filenames and specifying **-** for the standard input. The *files* will be printed in the order in which they appear on the command line. **lp** processes supplementary code set characters according to the locale specified in the **LC_CTYPE** environment variable [see **LANG** on **envvar**(BA_ENV)], except as noted under the **-t** option below.

The LP print service associates a unique *request-ID* with each request and displays it on the standard output. This *request-ID* can be used later when canceling or changing a request, or when determining its status. See the section on **cancel** for details about canceling a request, and **lpstat**(AU_CMD) for information about checking the status of a print request.

The second form of **lp** is used to change the options for a request submitted previously. The print request identified by the *request-ID* is changed according to the *print-options* specified with this command. The *print-options* available are the same as those with the first form of the **lp** command. If the request has finished printing, the change is rejected. If the request is already printing, it will be stopped and restarted from the beginning (unless the **-P** option has been given).

If a print job fails because of level range restrictions, the job will be canceled, and you will be notified by **mail**. In that case, you will need to submit the job to a different printer (one with the appropriate security level range). Ask your system administrator for information on printer security level ranges.

For printers configured to use the **B2** interface, unless you use the **-o nolabels** option, all paginated output will have a single line of security level information printed at the top and bottom of each page of the output. (The security level name is truncated if it is longer than one line.) In addition, the banner and trailer pages for the print job will contain complete security level information.

The **cancel** command allows users to cancel print requests previously sent with the **lp** command. The first form of **cancel** permits cancellation of requests based on their *request-ID*. The second form of **cancel** permits cancellation of requests based on the *login-ID* of their owner.

Sending a Print Request

The first form of the **lp** command is used to send a print request either to a particular printer or to any printer capable of meeting all requirements of the print request.

Options to **lp** must always precede filenames, but may be specified in any order. The following options are available for **lp**:

-c Make copies of the *files* to be printed immediately when **lp** is invoked. Normally *files* will not be copied, but will be linked whenever possible. If the **-c** option is not specified, the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that if the **-c** option is not specified, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.

-d *dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. If *dest* is **any**, then the request will be printed on any printer that can handle it. Under certain conditions (unavailability of printers, file space limitations, and so on) requests for specific destinations may not be accepted [see **lpstat**(AU_CMD)]. By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) **LPDEST**. If **LPDEST** is not set, then *dest* is taken from the environment variable **PRINTER**. If **PRINTER** is not set, a default destination (if one exists) for the computer system is used. Destination names vary between systems [see **lpstat**(AU_CMD)].

-f *form-name* [**-d any**]
Print the request on the form *form-name*. The LP print service ensures that the form is mounted on the printer. If *form-name* is requested with a printer destination that cannot support the form, the request is rejected. If *form-name* has not been defined for the system, or if the user is not allowed to use the form, the request is rejected. When the **-d any** option is given, the request is printed on any printer that has the requested form mounted and can handle all other needs of the print request.

-H *special-handling*
Print the request according to the value of *special-handling*. Acceptable values for *special-handling* are defined below:

hold Don't print the request until notified. If printing has already begun, stop it. Other print requests will go ahead of a held request until it is resumed. If the Auditing Utilities are installed, the use of this option is an auditable event.

resume Resume a held request. If it had been printing when held, it will be the next request printed, unless subsequently bumped by an **immediate** request. If the Auditing Utilities are installed, the use of this option is an auditable event. The **-i** option (followed by a *request-ID*) must be used whenever this argument is specified.

immediate (Available only to LP administrators)
Print the request next. If more than one request is assigned **immediate**, the most recent request will be printed first. If another request is currently printing, it must be put on hold to allow this immediate request to print.

-L *locale-name*
Specify *locale-name* as the locale to use with this print request. By default, *locale-name* is set to the value of **LC_CTYPE**. If **LC_CTYPE** is not set, *locale-name* defaults to the **C** locale.

-m Send mail [see **mail**(BU_CMD)] after the files have been printed. By default, mail is not sent upon normal completion of the print request.

-n *number* Print *number* copies of the output. The default is one copy.

-o *options* Specify printer-dependent *options*. Several such *options* may be collected by specifying the **-o** keyletter more than once (that is, **-o** $option_1$ **-o** $option_2$ . . . **-o** $option_n$), or by specifying a list of options with one **-o** keyletter enclosed in double quotes and separated by spaces (that is, **-o "***$option_1$ $option_2$ . . . $option_n$***"**).

nobanner Do not print a banner page with this request. The administrator can disallow this option at any time. This option is not supported by printers configured to use the **B2** interface.

nofilebreak
Do not insert a form feed between the files given, if submitting a job to print more than one file. This option is not supported by printers configured to use the **PS** (PostScript) interface.

nolabels Do not print security level information at the top and bottom of each page of the output. If the Auditing Utilities are installed, the use of this option is an auditable event. This option is not supported by printers configured to use the **standard** or **PS** (PostScript) interface.

length=*scaled-decimal-number*
Print this request with pages *scaled-decimal-number* long. A *scaled-decimal-number* is an optionally scaled decimal number that gives a size in lines, characters, inches, or centimeters, as appropriate. The scale is indicated by appending the letter **i** for inches, or the letter **c** for centimeters. For length or width settings, an unscaled number indicates lines or characters; for line pitch or character pitch settings, an unscaled number indicates lines per inch or characters per inch (the same as a number scaled with **i**). For example, **length=66** indicates a page length of 66 lines, **length=11i** indicates a page length of 11 inches, and **length=27.94c** indicates a

page length of 27.94 centimeters. This option may not be used with the **-f** option and is not supported by the **PS** (PostScript) or B2 interface.

width=*scaled-decimal-number*
Print this request with pages *scaled-decimal-number* wide. (See the explanation of *scaled-decimal-numbers* in the discussion of **length**, above.) This option may not be used with the **-f** option and is not supported by the **PS** (PostScript) or B2 interface.

lpi=*scaled-decimal-number*
Print this request with the line pitch set to *scaled-decimal-number*. (See the explanation of *scaled-decimal-numbers* in the discussion of **length**, above.) This option may not be used with the **-f** option and is not supported by the **PS** (PostScript) or B2 interface.

cpi=pica|elite|compressed
Print this request with the character pitch set to **pica** (representing 10 characters per inch), **elite** (representing 12 characters per inch), or **compressed** (representing as many characters per inch as a printer can handle). There is not a standard number of characters per inch for all printers; see the Terminfo database [**terminfo**] for the default character pitch for your printer. This option may not be used with the **-f** option and is not supported by the **PS** (PostScript) or B2 interface.

stty=*stty-option-list*
A list of options valid for the **stty** command; enclose the list with single quotes if it contains blanks.

-P *page-list* Print the pages specified in *page-list*. This option can be used only if there is a filter available to handle it; otherwise, the print request will be rejected. The *page-list* may consist of ranges of numbers, single page numbers, or a combination of both. The pages will be printed in ascending order.

-q *priority-level*
Assign this request *priority-level* in the printing queue. The values of *priority-level* range from 0 (highest priority) to 39 (lowest priority). If a priority is not specified, the default for the print service is used, as assigned by the system administrator. A priority limit may be assigned to individual users by the system administrator. If the Auditing Utilities are installed, the use of this option is an auditable event.

-r See ''**-T** *content-type* **[-r]**'' below.

-s Suppress the **request id is** . . . message.

-S *character-set* [**-d any**]
-S *print-wheel* [**-d any**]
Print this request using the specified *character-set* or *print-wheel*. If a form was requested and it requires a character set or print wheel other than the one specified with the **-S** option, the request is rejected.

For printers that take print wheels: if the print wheel specified is not one listed by the administrator as acceptable for the printer specified in this request, the request is rejected unless the print wheel is already mounted on the printer.

For printers that use selectable or programmable character sets: if the *character-set* specified is not one defined in the Terminfo database for the printer [see **terminfo**], or is not an alias defined by the administrator, the request is rejected.

When the **-d any** option is used, the request is printed on any printer that has the print wheel mounted or any printer that can select the character set, and that can handle all other needs of the request.

-t *title* Print *title* on the banner page of the output. The default is no title. Enclose *title* in quotes if it contains blanks. Supplementary code set characters specified in *title* are not printed correctly [see **banner**(BU_CMD)].

-T *content-type* **[-r**]
Print the request on a printer that can support the specified *content-type*. If no printer accepts this type directly, a filter will be used to convert the content into an acceptable type. If the **-r** option is specified, a filter will not be used. If **-r** is specified but no printer accepts the *content-type* directly, the request is rejected. If the *content-type* is not acceptable to any printer, either directly or with a filter, the request is rejected.

In addition to ensuring that no filters will be used, the **-r** option will force the equivalent of the **-o 'stty=-opost'** option.

-w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, or if the printer resides on a remote system, then mail will be sent instead. Be aware that messages may be sent to a window other than the one in which the command was originally entered.

-y *mode-list* Print this request according to the printing modes listed in *mode-list*. The allowed values for *mode-list* are locally defined. This option may be used only if there is a filter available to handle it; otherwise, the print request will be rejected.

The following list describes the *mode-list* options:

"**-y reverse**"
Reverse the order in which pages are printed.

"**-y landscape**"
Change the orientation of a physical page from portrait to landscape.

"**-y x=***number*,**y=***number*"
Change the default position of a logical page on a physical page by moving the origin.

"**-y group=***number*"
Group multiple logical pages on a single physical page.

"**-y magnify=***number*"
Change the logical size of each page in a document.

"**-o length=***number*"
Select the number of lines in each page of the document.

"**-P** *number*"
Select, by page numbers, a subset of a document to be printed.

"**-n** *number*"
Print multiple copies of a document.

Canceling a Print Request

The **cancel** command cancels requests for print jobs made with the **lp** command. The first form allows a user to specify one or more *request-IDs* of print jobs to be canceled. Alternatively, the user can specify one or more *printers*, on which only the currently printing job will be canceled if it is the user's job.

The second form of **cancel** cancels all jobs for users specified in *login-IDs*. In this form the *printers* option can be used to restrict the printers on which the users' jobs will be canceled. Note that in this form, when the *printers* option is used, all jobs queued by the users for those printers will be canceled. A printer class is not a valid argument.

A user without special privileges can cancel only requests that are associated with his or her own login ID; To cancel a request, a user issues the following command:

cancel -u *login-ID* [*printer*]

This command cancels all print requests associated with the *login-ID* of the user making the request, either on all printers (by default) or on the printer specified.

Administrative users with the appropriate privileges can cancel jobs submitted by any user by issuing the following types of commands:

cancel -u "*login-ID-list***"**
Cancels all requests (on all relevant printers) by the specified users, including those jobs currently being printed. Double quotes must be used around *login-ID-list* if the list contains blanks. The argument *login-ID-list* may include any or all of the following constructs:

login-ID	a user on the local system

system-name`!`*login-ID*	a user on system *system-name*
system-name`!all`	all users on system *system-name*
`all!`*login-ID*	a user on all systems
`all`	all users on the local system
`all!all`	all users on all systems

Note that a remote job can be canceled only if it originated on the client system; that is, a server system can cancel jobs that came from a client, and a client system can cancel jobs it sent to a server.

`cancel -u "`*login-ID-list***`"`** *printer-1 printer-2 printer-n*
Cancels all requests by the specified users for the specified printers, including those jobs currently being printed. (For a complete list of printers available on your system, execute the **`lpstat -p`** command.)

In any of these cases, the cancellation of a request that is currently printing frees the printer to print the next request.

If the Auditing Utilities are installed, the use of this command is an auditable event.

Downloading Type 1 PostScript Fonts to PostScript Printers

The desktop metaphor has a feature allowing the installation of retail Type 1 fonts for use with applications running under the metaphor. These fonts may be downloaded to PostScript printers if the application generates PostScript output that uses them. The **`lp`** command handles this automatically using the filter named **`download`**.

FILES

`/var/spool/lp/`*

`/usr/lib/locale/`*locale*`/LC_MESSAGES/uxlp`
language-specific message file [see **`LANG`** on **`envvar`**(BA_ENV)].

SEE ALSO

`lpstat`(AU_CMD), **`mail`**(BU_CMD)

LEVEL

Level 1.

NOTICES

Printers for which requests are not being accepted will not be considered when the destination is **`any`**. (Use the **`lpstat -a`** command to see which printers are accepting requests.) However, if a request is destined for a class of printers and the class itself is accepting requests, then all printers in the class will be considered, regardless of their acceptance status.

For printers that take mountable print wheels or font cartridges, if you do not specify a particular print wheel or font with the **`-S`** option, whichever one happens to be mounted at the time your request is printed will be used. The **`lpstat -p`** *printer* **`-l`** command is used to see which print wheels are available on a particular printer. The **`lpstat -S -l`** command is used to see what print wheels are available and on which printers. Without the **`-S`** option, the standard character set is used for printers that have selectable character sets.

If you experience problems with jobs that usually print but on occasion do not print, check the physical connections between the printer and your computer. If you are using an automatic data switch or an A/B switch, try removing it and see if the problem clears.

Pre-SVR4.2 systems may issue warnings about unrecognized options (such as the **locale=** or **flist=** options), when processing print requests from remote systems running a more recent version of the LP Print Server. The request will be printed normally, however.

Administrators with appropriate privileges can suppress these warnings by adding the following two lines to the section annotated as ''adding simple options,'' in the printer interface program used by the printer issuing the warnings.

```
locale=*) ;;
flist=*) ;;
```

(Printer interface programs are found in the **/usr/lib/lp/model** directory.) An example of how to do this can be found in the **standard** interface program.

NAME

lpstat – print information about the status of the LP print service

SYNOPSIS

`lpstat` [*options*]

DESCRIPTION

The `lpstat` command displays information about the current status of the LP print service. If no *options* are given, `lpstat` displays the status of all print requests made by you. [See lp(AU_CMD) for details.] If the command is issued on a system running the LP Network Utilities, `lpstat` displays the status of requests made to both local and remote printers. Status messages containing supplementary code set characters are displayed according to the locale specified in the `LC_CTYPE` environment variable [see `LANG` on envvar(BA_ENV)].

Any arguments that are not *options* are assumed to be *request-IDs* as returned by `lp`. The `lpstat` command displays the status of such requests. The *options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated by commas or a list of items separated by spaces (some form of quoting will be needed). For example:

`-p` *printer1*`,`*printer2*
`-u "`*user1 user2 user3*`"`

Specifying `all` after any keyletter that takes *list* as an argument causes all information relevant to the keyletter to be displayed. For example, the command

```
lpstat -o all
```

displays the status of all output requests.

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be displayed. For example, the command

```
lpstat -o
```

displays the status of all output requests.

If the Enhanced Security Extension is implemented and running, unprivileged users can display information only on requests that are associated with their own login IDs and that have a MAC level dominated by the users' current MAC level. Administrative users with the appropriate privileges may override these restrictions and report information on all jobs.

The following options and arguments may be used with `lpstat`:

`-a` [*list*]	Report whether print destinations are accepting requests. *list* is a list of intermixed printer names and class names.
`-c` [*list*]	Report names of all classes and their members. *list* is a list of class names.
`-d`	Report what the system default destination is (if any).
`-f` [*list*] [`-l`]	Verify that the forms in *list* are recognized by the LP print service. *list* is a list of forms; the default is `all`. The `-l` option will list the form parameters.

`-o` [*list*] [`-l`] Report the status of print requests. *list* is a list of intermixed printer names, class names, and *request-IDs*. The keyletter `-o` may be omitted. The `-l` option lists for each request whether it is queued for, assigned to, or being printed on a printer, the form required (if any), and the character set or print wheel required (if any).

`-p` [*list*] [`-D`] [`-l`] Report the status of printers. *list* is a list of printer names. If the `-D` option is given, a brief description is printed for each printer in *list*. If the `-l` option is given, a full description of each printer's configuration is given, including the form mounted, the acceptable content and printer types, a printer description, the interface used, and so on.

`-r` Report the status of the LP request scheduler (whether it is running).

`-R` Report a number showing the position of jobs in the print queue for each printer.

`-s` [`-l`] Display a status summary, including the status of the LP scheduler, the system default destination, a list of class names and their members, a list of printers and their associated devices, a list of the systems sharing print services, a list of all forms and their availability, and a list of all recognized character sets and print wheels. The `-l` option displays all parameters for each form and the printer name where each character set or print wheel is available.

`-S` [*list*] [`-l`] Verify that the character sets or the print wheels specified in *list* are recognized by the LP print service. Items in *list* can be character sets or print wheels; the default for *list* is `all`. If the `-l` option is given, each line is appended by a list of printers that can handle the print wheel or character set. The list also shows whether the print wheel or character set is mounted or specifies the built-in character set into which it maps.

`-t` [`-l`] Display all status information: all the information obtained with the `-s` option, plus the acceptance and idle/busy status of all printers and status of all requests. The `-l` option displays more detail as described for the `-f`, `-o`, `-p`, and `-s` options. Supplementary code set characters specified may not be printed correctly.

`-u` [*list*] Display the status of output requests for users. The *list* argument may include any or all of the following constructs:

login-ID	a user on any system
system-name!*login-ID*	a user on system *system-name*

system-name!`all`	all users on system *system-name*
`all!`*login-ID*	a user on all systems
`all`	all users on all systems

The default value of *list* is `all`.

`-v` [*list*] Report the names of printers and the pathnames of the devices associated with them (for local printers) or remote system names (if the LP Network Utilities are installed). Administrative users will also see the device level ranges of remote printers. *list* is a list of printer names.

If you select this option for a network printer, then, if possible, the specified printer range will be displayed by the security level aliases. If the level alias cannot be displayed, then the fully qualified level name will be displayed. If the fully qualified level name cannot be displayed, then the appropriate level ID number will be displayed.

`-z` Display the alias name of the MAC level associated with the print jobs; valid only if the Enhanced Security Extension is implemented.

`-Z` Display the fully qualified name of the MAC level associated with the print jobs; valid only if the Enhanced Security Extension is implemented.

The `-z` and `-Z` options are mutually exclusive. If the `-z` option is specified and there is not an alias assigned to the level, the decimal value of the level identifier (LID) is displayed. If the `-z` or `-Z` option is specified and the level is in the *valid-inactive* state, the decimal value of the LID is displayed. LID states are described in lvlname(ES_CMD).

FILES

`/etc/lp/*`
`/var/spool/lp/*`

SEE ALSO

lp(AU_CMD), lvlname(ES_CMD).

LEVEL

Level 1.

NAME

mailx – interactive message processing system

SYNOPSIS

`mailx [-e]`

`mailx [-f[`*filename*`]] [-H] [-N] [-u`*user*`] [-i] [-n]`

`mailx [-F] [-h`*number*`] [-r`*address*`] [-s`*subject*`] [-i] [-n]` *name ...*

DESCRIPTION

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

Incoming mail is stored in a standard file for each user, called the system mailbox for that user. When `mailx` is called to read messages, the mailbox is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the `mbox` and is normally located in the user's home directory (see `MBOX`, in **Environment Variables** below for a description of this file). Messages remain in this file until specifically removed.

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

Option for testing presence of mail:

`-e` Test for presence of mail. The command `mailx` prints nothing and exits with a successful return code if there is mail to read.

Options for receiving mail:

`-f[`*filename*`]`
: Read messages from *filename* instead of mailbox. If no *filename* is specified, the `mbox` is used.

`-H` Print header summary only.

`-N` Do not print initial header summary.

`-u`*user*
: Read *user*'s mailbox. This is only effective if *user*'s mailbox is not read protected.

Options for sending mail:

`-F` Record the message in a file named after the first recipient. Overrides the "record" variable, if set (see **Environment Variables**).

`-h`*number*
: The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops.

`-r`*address*
Pass *address* to network delivery software. All tilde commands are disabled.

`-s`*subject*
Set the `Subject` header field to *subject*.

Options for both sending and receiving mail:

`-i` Ignore interrupts. See also "ignore" (**Environment Variables**).

`-n` Do not initialize from the system default `mailx.rc` file.

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

At any time, the behavior of `mailx` is governed by a set of environment variables. These are flags and valued parameters which are set and cleared via the `set` and `unset` commands. See **Environment Variables** below for a summary of these parameters.

Regular commands are of the form

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

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

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

n Message number *n*.

`.` The current message.

`^` The first undeleted message.

`$` The last message.

`*` All messages.

n`-`*m* An inclusive range of message numbers.

user All messages from *user*.

/string All messages with *string* in the subject line (case ignored).

`:`*c* All messages of type *c*, where *c* is one of:

`d` deleted messages

`n` new messages

`o` old messages

`r` read messages

`u` unread messages

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

Other arguments are usually arbitrary strings whose usage depends on the command involved. Filenames, where expected, can be specified with metacharacters understood by the command interpreter. Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, `mailx` reads commands from a system-wide file to initialize certain parameters, then from a private start-up file (`$HOME/.mailrc`) for personalized variables. Most regular commands are legal inside start-up files, the most common use being to set up initial display options and alias lists. The following commands are not legal in the start-up file: `!`, `Copy`, `edit`, `followup`, `Followup`, `hold`, `mail`, `preserve`, `reply`, `Reply`, `shell`, and `visual`. Any errors in the start-up file cause the remaining lines in the file to be ignored.

Commands

The following is a complete list of `mailx` commands:

`!`*command*
Escape to the command interpreter. See `SHELL` (**Environment Variables**).

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

`=` Print the current message number.

`?` Prints a summary of commands.

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

`alt`ernates *name ...*
Declares a list of alternate names for the user's login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, `alt`ernates prints the current list of alternate names. See also `allnet` (**Environment Variables**).

`cd` [*directory*]
`ch`dir [*directory*]
Change directory. If *directory* is not specified, `$HOME` is used.

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

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

delete [*msglist*]
Delete messages from the mailbox. If `autoprint` is set, the next message after the last one deleted is printed (see **Environment Variables**).

`di`scard [*header-field ...*]
`ig`nore [*header-field ...*]
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are `status` and `cc`. The fields are included when the message is saved. The Print and Type commands override this command.

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

`ec`ho *string ...*
Echo the given *string*s [see echo(BU_CMD)].

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

`e`xit

`x`it Exit from `mailx`, without changing the mailbox. No messages are saved in the `mbox` (see also `q`uit).

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

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

Default file is the current mailbox.

`folders`
Print the names of the files in the directory set by the `folder` variable (see **Environment Variables**).

`f`ollowup [*message*]
Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the `record` variable, if set. See also the Followup, Save, and **C**opy commands and `outfolder`

(**Environment Variables**).

Followup [*msglist*]
: Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the followup, Save, and **C**opy commands and outfolder (**Environment Variables**).

from [*msglist*]
: Prints the header summary for the specified messages.

group *alias name ...*
alias *alias name ...*
: Declare an alias for the given *names*. The names will be substituted when *alias* is used as a recipient. Useful in the .mailrc file.

headers [*message*]
: Prints the page of headers which includes the message specified. The screen variable sets the number of headers per page (see **Environment Variables**). See also the z command.

help Prints a summary of commands.

hold [*msglist*]
preserve [*msglist*]
: Holds the specified messages in the mailbox.

if s|r
: *mail-command*s

else
: *mail-command*s

endif Conditional execution, where s will execute *mail-command*s, up to an else or endif, if the program is in send mode; r causes the *mail-command*s to be executed only in receive mode. Useful in the .mailrc file.

ignore *header-field ...*
discard *header-field ...*
: Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are status and cc. All fields are included when the message is saved. The Print and Type commands override this command.

list Prints all commands available. No explanation is given.

mail *name ...*
: Mail a message to the specified users.

mbox [*msglist*]
: Arrange for the given messages to end up in the standard mbox save file when mailx terminates normally. See MBOX (**Environment Variables**) for a description of this file. See also the exit and quit commands.

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

pipe [*msglist*] [*command*]
| [*msglist*] [*command*]
: Pipe the message through the given *command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the `cmd` variable. If the `page` variable is set, a form feed character is inserted after each message (see **Environment Variables**).

preserve [*msglist*]
hold [*msglist*]
: Preserve the specified messages in the mailbox.

Print [*msglist*]
Type [*msglist*]
: Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the `ignore` command.

print [*msglist*]
type [*msglist*]
: Print the specified messages. If `crt` is set, the messages longer than the number of lines specified by the `crt` variable are paged through the command specified by the `PAGER` environment variable. The default command is `pg`. (See **Environment Variables**).

quit
: Exit from `mailx`, storing messages that were read in `mbox` and unread messages in the mailbox. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]
Respond [*msglist*]
: Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If `record` is set to a filename, the response is saved at the end of that file (see **Environment Variables**).

reply [*message*]
respond [*message*]
: Reply to the specified message, including all other recipients of the message. If `record` is set to a filename, the response is saved at the end of that file (see **Environment Variables**).

Save [*msglist*]
: Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the **C**opy, `f`ollowup, and Followup commands and `outfolder` (**Environment Variables**).

save [*filename*]
save [*msglist*] *filename*
: Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the mailbox when `mailx` terminates unless `keepsave` is set (see also **Environment Variables** and the `exit` and `quit` commands).

set
set *name*
set *name=string*
set *name=number*
: Define a variable called *name*. The variable may be given a null, string, or numeric value. `set` by itself prints all defined variables and their values. See **Environment Variables** for detailed descriptions of the `mailx` variables.

shell
: Invoke an interactive command interpreter (see also `SHELL` (**Environment Variables**)).

size [*msglist*]
: Print the size in characters of the specified messages.

source *filename*
: Read commands from the given file and return to command mode.

top [*msglist*]
: Print the top few lines of the specified messages. If the `toplines` variable is set, it is taken as the number of lines to print (see **Environment Variables**). The default is 5.

touch [*msglist*]
: Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the `mbox` upon normal termination. See `exit` and `quit`.

Type [*msglist*]
Print [*msglist*]
: Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the `ignore` command. If `crt` is set, the messages longer than the number of lines specified by the `crt` variable are paged through the command specified by the `PAGER` variable. The default command is `pg`. (See **Environment Variables**).

type [*msglist*]
print [*msglist*]
: Print the specified messages. If `crt` is set, the messages longer than the number of lines specified by the `crt` variable are paged through the command specified by the `PAGER` variable. The default command is `pg`. (See **Environment Variables**).

undelete [*msglist*]
: Restore the specified deleted messages. Only messages deleted in the current mail session will be restored. If `autoprint` is set, the last message of those restored is printed (see **Environment Variables**).

`unset` *name ...*
Causes the specified variables to be erased. If the variable was imported from the execution environment (*i.e.*, an environment variable) then it cannot be erased.

`version`
Prints the current version and release date.

`visual` [*msglist*]
Edit the given messages with a screen editor. The messages are placed in a temporary file and the `VISUAL` variable is used to get the name of the editor (see **Environment Variables**).

`write` [*msglist*] *filename*
Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the `save` command.

`xit`

`exit` Exit from `mailx`, without changing the mailbox. No messages are saved in the `mbox` (see also `quit`).

`z`[`+`|`-`]
Scroll the header display forward or backward one screen full. The number of headers displayed is set by the `screen` variable (see **Environment Variables**).

Tilde Escapes

The following commands may be entered only from input mode, by beginning a line with the tilde escape character (`~`). See `escape` (**Environment Variables**) for changing this special character.

`~!` *command*
Escape to the command interpreter.

`~.` Simulate end of file (terminate message input).

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

`~?` Print a summary of tilde escapes.

`~A` Insert the autograph string `Sign` into the message (see **Environment Variables**).

`~a` Insert the autograph string `sign` into the message (see **Environment Variables**).

`~b` *name ...*
Add the *name*s to the blind carbon copy (Bcc) list.

`~c` *name ...*
Add the *name*s to the carbon copy (Cc) list.

`~d` Read in the `dead.letter` file. See `DEAD` (**Environment Variables**) for a description of this file.

`~e` Invoke the editor on the partial message. See also `EDITOR` (**Environment Variables**).

`~f` [*msglist*]
Forward the specified messages. The messages are inserted into the message, without alteration. This command is valid only when sending a message while reading mail.

`~h` Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if it had just been typed.

`~i` *string*
Insert the value of the named variable into the text of the message. For example, `~A` is equivalent to '`~i Sign`'.

`~m` [*msglist*]
Insert the specified messages into the letter, shifting the new text to the right one tab stop. This command is valid only when sending a message while reading mail.

`~p` Print the message being entered.

`~q` Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in `dead.letter`. See `DEAD` (**Environment Variables**) for a description of this file.

`~r` *filename*

`~<`*filename*

`~<!`*command*
Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary system command and is executed, with the standard output inserted into the message.

`~s` *string ...*
Set the subject line to *string*.

`~t` *name ...*
Add the given *name*s to the To list.

`~v` Invoke a preferred screen editor on the partial message. See also `VISUAL` (**Environment Variables**).

`~w` *filename*
Write the partial message onto the given file, without the header.

`~x` Exit as with `~q` except the message is not saved in `dead.letter`.

`~|` *command*
Pipe the body of the message through the given *command*. If the *command* returns a successful exit status, the output of the command replaces the message.

Environment Variables

The following are environment variables taken from the execution environment and are not alterable within `mailx`.

`HOME`=*directory*
The user's base of operations.

`MAILRC`=*filename*
The name of the start-up file. Default is `$HOME/.mailrc`.

The following variables are internal `mailx` variables. They may be imported from the execution environment or set via the `set` command at any time. The `unset` command may be used to erase variables.

`allnet`
All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is `noallnet`. See also the `alternates` command and the `metoo` variable.

`append`
Upon termination, append messages to the end of the `mbox` file instead of prepending them. Default is `noappend`.

`askcc`
Prompt for the Cc list after message is entered. Default is `noaskcc`.

`asksub`
Prompt for subject if it is not specified on the command line with the `-s` option. Enabled by default.

`autoprint`
Enable automatic printing of messages after `delete` and `undelete` commands. Default is `noautoprint`.

`bang` Enable the special-case treatment of exclamation points (!) in escape com-

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

`EDITOR`=*command*
The command to run when the `e`dit or `~e` command is used. Default is `ed`.

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

`folder`=*directory*
The directory for saving standard mail files. Filenames used with the `copy`, `folder`, `send`, and `write` commands, that begin with a plus (+), are expanded by preceding the filename with this directory name to obtain the real filename. If *directory* does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a `mailx` command line, `folder` must be an exported environment variable. There is no default for the `folder` variable. See also `outfolder` below.

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

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

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

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

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

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

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

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

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

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

outfolder
Causes the files used to record outgoing messages to be located in the directory specified by the folder variable unless the pathname is absolute. Default is nooutfolder. See folder above and the Save, **C**opy, followup, and Followup commands.

page Used with the pipe command to insert a form feed after each message sent through the pipe. Default is nopage.

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

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

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

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

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

screen=*number*
Sets the number of lines in a screen full of headers for the headers command.

sendmail=*command*
Alternate command for delivering messages. Default is mail.

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

SHELL=*command*
The name of a preferred command interpreter. Default is sh.

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

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

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

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

`VISUAL`=*command*
The name of a preferred screen editor. Default is `vi`.

FILES

`$HOME/.mailrc`	user's start-up file
`$HOME/mbox`	secondary storage file

USAGE

End-user.

SEE ALSO

ed(BU_CMD), mail(BU_CMD), pg(BU_CMD), ls(BU_CMD), vi(AU_CMD).

LEVEL

Level 1.

NAME

mesg – permit or deny messages

SYNOPSIS

`mesg [y|n]`

DESCRIPTION

The command `mesg` with argument `n` prevents another user from writing to the invoking user's terminal, (*e.g.*, by using `write` [see write(AU_CMD)]). The command `mesg` with argument `y` reinstates write permission. With no arguments, `mesg` reports the current state without changing it.

ERRORS

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

SEE ALSO

write(AU_CMD).

LEVEL

Level 1.

NAME

newgrp – change to a new group

SYNOPSIS

`newgrp` [–] [*group*]

DESCRIPTION

The command `newgrp` changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs.

Exported environment variables retain their values after invoking `newgrp`; however, all unexported variables are either reset to their default value or set to `null`. Environment variables (such as `PS1`, `PS2`, `PATH`, `MAIL`, and `HOME`), unless exported, are reset to default values.

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

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

FILES

`/etc/group` system's group file

`/etc/passwd` system's password file

USAGE

End-user.

SEE ALSO

sh(BU_CMD).

LEVEL

Level 1.

NAME

news – print news items

SYNOPSIS

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

DESCRIPTION

The command `news` prints files from the system news directory.

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

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

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

The `-s` option causes `news` to report how many current items exist, without printing their names or contents, and without changing the stored time.

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

If an interrupt (`DEL` or `BREAK`) is typed during the printing of a news item, printing stops and the next item is started. Another interrupt within one second of the first causes the program to terminate.

FILES

`$HOME/.news_time`

USAGE

End-user.

LEVEL

Level 1.

NAME

od – octal dump

SYNOPSIS

od [-bcDdFfOoSsvXx] [*file*] [[+]*offset*[.][b][x]]

DESCRIPTION

The command od prints *file* in one or more formats as selected by the options. If no file is specified, the standard input is used. If no option is specified, -o is the default.

For the purposes of this description, *word* refers to a 16-bit unit, independent of the word size of the machine.

The meanings of the options are:

-b Interpret bytes in octal.

-c Interpret bytes as single byte characters. Certain non-graphic characters appear as C language escapes, e.g., NUL=\0, BS=\b, FF=\f, NL=\n, CR=\r, HT=\t; others appear as 3-digit octal numbers. Multibyte characters are treated as non-graphic characters.

-D Interpret long words as unsigned decimal.

-d Interpret *words* in unsigned decimal.

-F Interpret double long words as extended precision.

-f Interpret long words as floating point.

-O Interpret long words as unsigned octal.

-o Interpret *words* in octal.

-S Interpret long words as signed decimal.

-s Interpret *words* in signed decimal.

-v Show all data (verbose).

-X Interpret long words in hexadecimal.

-x Interpret *word*s in hexadecimal.

The *offset* argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If . is appended, the offset is interpreted in decimal. If b is appended, the offset is interpreted in units of 512 bytes. If x is appended, offset is interpreted in hexadecimal. If offset is omitted, the . and **x** can still be used for displaying decimal and hexadecimal addresses respectively. If the *file* argument is omitted, the *offset* argument must be preceded by +.

LEVEL

Level 1.

NAME

passwd – change login password

SYNOPSIS

passwd [*name*]

DESCRIPTION

The command **passwd** changes or installs a password associated with the login *name*.

Ordinary users may change only the password which corresponds to their login *name*.

The command **passwd** prompts ordinary users for their old password, if any. It then prompts for the new password twice. If password aging is in effect, then the first time the new password is entered, **passwd** checks to see if the old password has ''aged'' sufficiently. If ''aging'' is insufficient the new password is rejected and **passwd** terminates.

If ''aging'' is sufficient, a check is made to insure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times.

A user with appropriate privileges may change any password; hence, **passwd** does not prompt such users for the old password. A user with appropriate privileges is not forced to comply with password aging and password construction requirements. Such users can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd

USAGE

End-user.

LEVEL

Level 1.

NAME

priocntl – process scheduler control

SYNOPSIS

priocntl -l

priocntl -d [-i *idtype***] [***idlist***]**

priocntl -s [-c *class***] [***class-specific options***] [-i** *idtype***] [***idlist***]**

priocntl -e [-c *class***] [***class-specific options***]** *command* **[***argument(s)***]**

DESCRIPTION

The **priocntl** command displays or sets scheduling parameters of the specified process(es). It can also be used to display the current configuration information for the system's process scheduler or execute a command with specified scheduling parameters.

Processes fall into distinct classes with a separate scheduling policy applied to each class. The two process classes currently supported are the fixed priority class and the time-sharing class. The characteristics of these two classes and the class-specific options they accept are described below under the headings FIXED PRIORITY CLASS and TIME-SHARING CLASS. With appropriate permissions, the **priocntl** command can change the class and other scheduling parameters associated with a running process.

In the default configuration, a runnable fixed priority process runs before any other process. Therefore, inappropriate use of fixed priority processes can have a dramatic negative impact on system performance.

The command

priocntl -l

displays a list of classes currently configured in the system along with class-specific information about each class. The format of the class-specific information displayed is described under the appropriate heading below.

The **-d** and **-s** options to **priocntl** allow the user to display or set the scheduling parameters associated with a set of processes. The **-i** option and its associated *idtype* argument, together with the *idlist* arguments to **priocntl** (if any), specify one or more processes to which the **priocntl** command is to apply. The interpretation of *idlist* depends on the value of *idtype*. The valid *idtype* arguments and corresponding interpretations of *idlist* are as follows:

-i pid *idlist* is a list of process IDs. The **priocntl** command applies to the specified processes.

-i ppid *idlist* is a list of parent process IDs. The **priocntl** command applies to all processes whose parent process ID is in the list.

-i pgid *idlist* is a list of process group IDs. The **priocntl** command applies to all processes in the specified process groups.

-i sid *idlist* is a list of session IDs. The **priocntl** command applies to all processes in the specified sessions.

-i class *idlist* consists of a single class name (**FP** for fixed priority or **TS** for time-sharing). The **priocntl** command applies to all processes in the specified class.

-i uid *idlist* is a list of user IDs. The **priocntl** command applies to all processes with an effective user ID equal to an ID from the list.

-i gid *idlist* is a list of group IDs. The **priocntl** command applies to all processes with an effective group ID equal to an ID from the list.

-i all The **priocntl** command applies to all existing processes. No *idlist* should be specified (if one is it is ignored). The permission restrictions described below still apply.

If the **-i** *idtype* option is omitted when using the **-d** or **-s** options the default *idtype* of **pid** is assumed.

If an *idlist* is present it must appear last on the command line and the elements of the list must be separated by white space. If no *idlist* is present an *idtype* argument of **pid**, **ppid**, **pgid**, **sid**, **class**, **uid**, or **gid** specifies the process ID, parent process ID, process group ID, session ID, class, user ID, or group ID respectively of the **priocntl** command itself.

The command

priocntl -d [-i *idtype***] [***idlist***]**

displays the class and class-specific scheduling parameters of the process(es) specified by *idtype* and *idlist*.

The command

priocntl -s [-c *class***] [***class-specific options***] [-i** *idtype***] [***idlist***]**

sets the class and class-specific parameters of the specified processes to the values given on the command line. The **-c** *class* option specifies the class to be set. (The valid *class* arguments are **FP** for fixed priority or **TS** for time-sharing). The class-specific parameters to be set are specified by the class-specific options as explained under the appropriate heading below. If the **-c** *class* option is omitted, *idtype* and *idlist* must specify a set of processes that are all in the same class, otherwise an error results. If no class-specific options are specified the process's class-specific parameters are set to the default values for the class specified by **-c** *class* (or to the default parameter values for the process's current class if the **-c** *class* option is also omitted).

To change the scheduling parameters of a process using **priocntl** the real or effective user ID of the user invoking **priocntl** must match the real or effective user ID of the receiving process the user must be a privileged user. These are the minimum permission requirements enforced for all classes. An individual class may impose additional permissions requirements when setting processes to that class or when setting class-specific scheduling parameters.

When *idtype* and *idlist* specify a set of processes, **priocntl** acts on the processes in the set in an implementation-specific order. If **priocntl** encounters an error for one or more of the target processes, it may or may not continue through the set of processes, depending on the error. If the error is related to permissions, **priocntl** prints an error message and then continues through the process set, resetting the

parameters for all target processes for which the user has appropriate permissions. If **priocntl** encounters an error other than permissions, it does not continue through the process set but prints an error message and exits immediately.

A special **sys** scheduling class exists for scheduling the execution of certain special system processes (such as the swapper process). It is not possible to change the class of any process to **sys**. In addition, any processes in the **sys** class that are included in the set of processes specified by *idtype* and *idlist* are disregarded by **priocntl**. For example, if *idtype* were **uid**, an *idlist* consisting of a zero would specify all processes with a UID of zero except processes in the **sys** class and (if changing the parameters using the **-s** option) the **init** process.

The **init** process may be assigned to any class configured on the system; but the time-sharing class is almost always the appropriate choice. (Other choices may be highly undesirable; see your system administration documentation for more information.)

The command

priocntl -e [-c *class***] [***class-specific options***]** *command* **[***argument***(***s***)]**

executes the specified command with the class and scheduling parameters specified on the command line (*arguments* are the arguments to the *command*). If the **-c** *class* option is omitted the command is run in the user's current class.

FIXED PRIORITY CLASS

The fixed priority class provides a fixed priority preemptive scheduling policy for those processes requiring fast and deterministic response and absolute user/application control of scheduling priorities. If the fixed priority class is configured in the system it should have exclusive control of the highest range of scheduling priorities on the system. This ensures that a runnable fixed priority process is given CPU service before any process belonging to any other class.

The fixed priority class has a range of fixed priority (*fppri*) values that may be assigned to processes within the class. Fixed priority priorities range from 0 to *x*, where the value of *x* is configurable and can be displayed for a specific installation by using the command

priocntl -l

The fixed priority scheduling policy is a fixed priority policy. The scheduling priority of a fixed priority process never changes except as the result of an explicit request by the user/application to change the *fppri* value of the process.

For processes in the fixed priority class, the *fppri* value is, for all practical purposes, equivalent to the scheduling priority of the process. The *fppri* value completely determines the scheduling priority of a fixed priority process relative to other processes within its class. Numerically higher *fppri* values represent higher priorities. Since the fixed priority class controls the highest range of scheduling priorities in the system it is guaranteed that the runnable fixed priority process with the highest *fppri* value is always selected to run before any other process in the system.

In addition to providing control over priority, **priocntl** provides for control over the length of the time quantum allotted to processes in the fixed priority class. The time quantum value specifies the maximum amount of time a process may run assuming that it does not complete or enter a resource or event wait state (**sleep**).

Note that if another process becomes runnable at a higher priority the currently running process may be preempted before receiving its full time quantum.

The command

priocntl -d [-i *idtype***] [***idlist***]**

displays the fixed priority and time quantum (in millisecond resolution) for each fixed priority process in the set specified by *idtype* and *idlist*.

The valid class-specific options for setting fixed priority parameters are:

-p *fppri*	Set the fixed priority of the specified process(es) to *fppri*.
-t *tqntm* **[-r** *res***]**	Set the time quantum of the specified process(es) to *tqntm*. You may optionally specify a resolution as explained below.

Any combination of the **-p** and **-t** options may be used with **priocntl -s** or **priocntl -e** for the fixed priority class. If an option is omitted and the process is currently fixed priority the associated parameter is unaffected. If an option is omitted when changing the class of a process to fixed priority from some other class, the associated parameter is set to a default value. The default value for *fppri* is 0 and the default for time quantum is dependent on the value of *fppri* and on the system configuration.

When using the **-t** *tqntm* option you may optionally specify a resolution using the **-r** *res* option. (If no resolution is specified, millisecond resolution is assumed.) If *res* is specified it must be a positive integer between 1 and 1,000,000,000 inclusive and the resolution used is the reciprocal of *res* in seconds. For example, specifying **-t 10 –r 100** would set the resolution to hundredths of a second and the resulting time quantum length would be 10/100 seconds (one tenth of a second). Although very fine (nanosecond) resolution may be specified, the time quantum length is rounded up by the system to the next integral multiple of the system clock's resolution. The system clock's resolution is hardware-dependent; this resolution can be calculated from the value of **HZ**, which is defined in the file **/usr/include/sys/param.h**. **HZ** gives the number of clock ticks per second of the system clock. For example, an **HZ** of 100 specifies 100 clock ticks per second, or one tick every 10 milliseconds (that is, this system clock has a resolution of 10 milliseconds). If the **-t** and **-r** options are used to specify a time quantum of 34 milliseconds, it is rounded up to 4 ticks (40 milliseconds) on a machine with an **HZ** of 100. Requests for time quantums of zero or quantums greater than the (typically very large) implementation-specific maximum quantum result in an error.

To change the class of a process to fixed priority (from any other class) the user invoking **priocntl** must have appropriate privilege. To change the *fppri* value or time quantum of a fixed priority process the user invoking **priocntl** must either be a privileged user, or must currently be in the fixed priority class (shell running as a fixed priority process) with a real or effective user ID matching the real or effective user ID of the target process.

The fixed priority and time quantum are inherited across the **fork**(BA_OS) and **exec**(BA_OS) system calls.

EXAMPLES

priocntl -s -c FP -t 1 -r 10 -i *idtype idlist*

sets the class of any non-fixed priority processes selected by *idtype* and *idlist* to fixed priority and sets their fixed priority to the default value of 0. The fixed priority priorities of any processes currently in the fixed priority class are unaffected. The time quantums of all the specified processes are set to 1/10 seconds.

priocntl -e -c FP -p 15 -t 20 *command*

executes *command* in the fixed priority class with a fixed priority of 15 and a time quantum of 20 milliseconds.

TIME-SHARING CLASS

The time-sharing scheduling policy provides for a fair and effective allocation of the CPU resource among processes with varying CPU consumption characteristics. The objectives of the time-sharing policy are to provide good response time to interactive processes and good throughput to CPU-bound jobs while providing a degree of user/application control over scheduling.

The time-sharing class has a range of time-sharing user priority (*tsupri*) values that may be assigned to processes within the class. User priorities range from *–x* to *+x*, where the value of *x* is configurable. The range for a specific installation can be displayed by using the command

priocntl -l

The purpose of the user priority is to provide some degree of user/application control over the scheduling of processes in the time-sharing class. Raising or lowering the *tsupri* value of a process in the time-sharing class raises or lowers the scheduling priority of the process. It is not guaranteed, however, that a time-sharing process with a higher *tsupri* value will run before one with a lower *tsupri* value. This is because the *tsupri* value is just one factor used to determine the scheduling priority of a time-sharing process. The system may dynamically adjust the internal scheduling priority of a time-sharing process based on other factors such as recent CPU usage.

In addition to the system-wide limits on user priority (displayed with **priocntl -l**), there is a per process user priority limit (*tsuprilim*), which specifies the maximum *tsupri* value that may be set for a given process.

The command

priocntl -d [-i *idtype***] [***idlist***]**

displays the user priority and user priority limit for each time-sharing process in the set specified by *idtype* and *idlist*.

The valid class-specific options for setting time-sharing parameters are:

-m *tsuprilim*	Set the user priority limit of the specified process(es) to *tsuprilim*.
-p *tsupri*	Set the user priority of the specified process(es) to *tsupri*.

Any time-sharing process may lower its own *tsuprilim* (or that of another process with the same user ID). Only a time-sharing process with appropriate privilege may raise a *tsuprilim*. When changing the class of a process to time-sharing from some other class, appropriate privilege is required to set the initial *tsuprilim* to a value greater than zero.

Any time-sharing process may set its own *tsupri* (or that of another process with the same user ID) to any value less than or equal to the process's *tsuprilim*. Attempts to set the *tsupri* above the *tsuprilim* (and/or set the *tsuprilim* below the *tsupri*) result in the *tsupri* being set equal to the *tsuprilim*.

Any combination of the **-m** and **-p** options may be used with **priocntl -s** or **priocntl -e** for the time-sharing class. If an option is omitted and the process is currently time-sharing the associated parameter is normally unaffected. The exception is when the **-p** option is omitted and **-m** is used to set a *tsuprilim* below the current *tsupri*. In this case the *tsupri* is set equal to the *tsuprilim* which is being set. If an option is omitted when changing the class of a process to time-sharing from some other class, the associated parameter is set to a default value. The default value for *tsuprilim* is 0 and the default for *tsupri* is to set it equal to the *tsuprilim* value that is being set.

The time-sharing user priority and user priority limit are inherited across the **fork**(BA_OS) and **exec**(BA_OS) system calls.

EXAMPLES

priocntl -s -c TS -i *idtype idlist*

sets the class of any non-time-sharing processes selected by *idtype* and *idlist* to time-sharing and sets both their user priority limit and user priority to 0. Processes already in the time-sharing class are unaffected.

priocntl -e -c TS -m 0 -p -15 *command* **[***arguments***]**

executes *command* with the arguments *arguments* in the time-sharing class with a user priority limit of 0 and a user priority of –15.

VC CLASS

The idtype -i class is designated Level 2, June 1993.

NAME

stty – set the options for a terminal

SYNOPSIS

`stty` [`–a`] [`–g`] [*options*]

DESCRIPTION

The command `stty` sets certain terminal I/O options for the device that is its standard input; without arguments, it reports the settings of certain options; with the `–a` option, it reports all of the option settings; with the `–g` option, it reports current settings in a form that can be used as an argument to another `stty` command. For detailed information about the modes listed from **Control Modes** through **Local Modes**, below, see termio(BA_DEV). For detailed information about the modes listed under **Hardware Flow Control Modes** and **Clock Modes**, below, see termiox(BA_DEV). Options in the **Combination Modes** group are implemented using options in the groups listed before it. Note that many combinations of options make no sense, but no sanity checking is performed. *options* are selected from the following:

Control Modes

`parenb` (`–parenb`)
enable (disable) parity generation and detection.

`parext` (`–parext`)
enable (disable) extended parity generation and detection for mark and space parity.

`parodd` (`–parodd`)
select odd (even) parity, or mark (space) parity if `parext` is enabled.

`cs5 cs6 cs7 cs8`
select character size.

`0` the modem control lines will no longer be asserted. Usually this will disconnect the line.

number
set terminal baud rate to the *number* given, if possible (all speeds are not supported by all hardware interfaces).

`ispeed` *number*
set terminal baud rate to the *number* given, if possible. (Not all hardware supports split baud rates.) If the baud rate is set to zero, the input baud rate will be specified by the value of the output baud rate.

`ospeed` *number*
set terminal output baud rate to the *number* given, if possible (Not all hardware supports split baud rates.) If the baud rate is set to zero, the modem control lines will no longer be asserted. Usually this will disconnect the line.

`hupcl` (`–hupcl`)
hang up (do not hang up) modem connection on last close.

`hup` (`–hup`)
same as `hupcl` (`–hupcl`).

`cstopb` (`–cstopb`)
use two (one) stop bits per character.

`cread` (`–cread`)
enable (disable) the receiver.

`clocal` (`–clocal`)
assume a line without (with) modem control.

Input Modes

`ignbrk` (`–ignbrk`)
ignore (do not ignore) break on input.

`brkint` (`–brkint`)
signal (do not signal) `INTR` on break.

`ignpar` (`–ignpar`)
ignore (do not ignore) parity errors.

`parmrk` (`–parmrk`)
mark (do not mark) parity errors.

`inpck` (`–inpck`)
enable (disable) input parity checking.

`istrip` (`–istrip`)
strip (do not strip) input characters to seven bits.

`inlcr` (`–inlcr`)
map (do not map) `NL` to `CR` on input.

`igncr` (`–igncr`)
ignore (do not ignore) `CR` on input.

`icrnl` (`–icrnl`)
map (do not map) `CR` to `NL` on input.

`iuclc` (`–iuclc`)
map (do not map) upper-case alphabetics to lower case on input.

`ixon` (`–ixon`)
enable (disable) `START`/`STOP` output control. Output is stopped by sending an ASCII `DC3` and started by sending an ASCII `DC1`.

`ixany` (`–ixany`)
allow any character (only `DC1`) to restart output.

`ixoff` (`–ixoff`)
request that the system send (not send) `START`/`STOP` characters when the input queue is nearly empty/full.

`imaxbel` (`–imaxbel`)
echo (do not echo) `BEL` when the input line is too long.

Output Modes

`opost` (`–opost`)
post-process output (do not post-process output; ignore all other output modes).

`olcuc` (`–olcuc`)
map (do not map) lower-case alphabetics to upper case on output.

`onlcr` (`–onlcr`)
map (do not map) `NL` to `CR-NL` on output.

`ocrnl` (`–ocrnl`)
map (do not map) `CR` to `NL` on output.

`onocr` (`–onocr`)
do not (do) output `CR`s at column zero.

`onlret` (`–onlret`)
on the terminal `NL` performs (does not perform) the `CR` function.

`ofill` (`–ofill`)
use fill characters (use timing) for delays.

`cr0 cr1 cr2 cr3`
select style of delay for carriage returns.

`nl0 nl1`
select style of delay for linefeeds.

`tab0 tab1 tab2 tab3`
select style of delay for horizontal tabs.

`bs0 bs1`
select style of delay for backspaces.

`ff0 ff1`
select style of delay for form-feeds.

`vt0 vt1`
select style of delay for vertical tabs.

Local Modes

`isig` (`–isig`)
enable (disable) the checking of characters against the special control characters `INTR`, `QUIT`, and `SWTCH`.

`icanon` (`–icanon`)
enable (disable) canonical input (`ERASE` and `KILL` processing).

`xcase` (`–xcase`)
canonical (unprocessed) upper / lower-case presentation.

`echo` (`–echo`)
echo back (do not echo back) every character typed.

`echoe` (`–echoe`)
echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be

confusing on escaped characters, tabs, and backspaces.

`echok` (`–echok`)
echo (do not echo) `NL` after `KILL` character.

`lfkc` (`–lfkc`)
the same as `echok` (`–echok`); obsolete.

`echonl` (`–echonl`)
echo (do not echo) `NL` when `echo` is disabled.

`noflsh` (`–noflsh`)
disable (enable) flush after `INTR`, `QUIT`, or `SWTCH`.

`tostop` (`–tostop`)
send (do not send) `SIGTTOU` for background processes.

`echoctl` (`–echoctl`)
echo (do not echo) control characters as ^*char*, delete as ^?.

`echoprt` (`–echoprt`)
echo (do not echo) erase character as character is erased.

`echoke` (`–echoke`)
BS-SP-BS erase (do not BS-SP-BS erase) entire line on line kill.

`flusho` (`–flusho`)
Output is (is not) being flushed.

`pendin` (`–pendin`)
Retype (do not retype) pending input at next read or input character.

`iexten` (`–iexten`)
enable (disable) extended (implementation-defined) functions for input data.

Hardware Flow Control Modes

The following options are reserved for use with those devices that support hardware flow control. If the functionality is supported, then this interface must be used.

`rtsxoff` (`–rtsxoff`)
enable (disable) RTS hardware flow control on input.

`ctsxon` (`–ctsxon`)
enable (disable) CTS hardware flow control on output.

`dtrxoff` (`–dtrxoff`)
enable (disable) DTR hardware flow control on input.

`cdxon` (`–cdxon`)
enable (disable) CD hardware flow control on output.

`isxoff` (`–isxoff`)
enable (disable) isochronous hardware flow control on input.

Clock Modes

The following options are reserved for use with those devices that support clock modes. If the functionality is supported, then this interface must be used.

`xcibrg` get transmit clock from internal baud rate generator.

`xctset` get the transmit clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.

`xcrset` get transmit clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.

`rcibrg` get receive clock from internal baud rate generator.

`rctset` get receive clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.

`rcrset` get receive clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.

`tsetcoff` transmitter signal element timing clock not provided.

`tsetcrbrg` output receive baud rate generator on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.

`tsetctbrg` output transmit baud rate generator on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.

`tsetctset` output transmitter signal element timing (DCE source) on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.

`tsetcrset` output receiver signal element timing (DCE source) on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.

`rsetcoff` receiver signal element timing clock not provided.

`rsetcrbrg` output receive baud rate generator on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.

`rsetctbrg` output transmit baud rate generator on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.

`rsetctset` output transmitter signal element timing (DCE source) on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.

`rsetcrset` output receiver signal element timing (DCE source) on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.

Control Assignments

control-character c

set *control-character* to *c*, where *control-character* is one of: `erase`, `kill`, `intr`, `quit`, `eof`, `eol`, `eol2`, `start`, `stop`, `susp`, `dsusp`, `werase`, or `lnext`. If *c* is preceded by a caret (`^`), then the value used is the corresponding `CTRL` character (*e.g.*, ''`^d`'' is a `CTRL-d`); ''`^?`'' is interpreted as `DEL` and ''`^-`'' is interpreted as undefined.

`min, time` *number*
Set the value of `min` or `time` to *number*. `min` and `time` are used in non-Canonical mode input processing (`-icanon`).

`line` *i* set line discipline to *i* ($0 < i < 127$).

Combination Modes

`evenp` or `parity`
enable `parenb` and `cs7`.

`oddp` enable `parenb`, `cs7`, and `parodd`.

`spacep` enable `parenb`, `cs7`, and `parext`.

`markp` enable `parenb`, `cs7`, `parodd`, and `parext`.

`-parity` or `-evenp`
disable `parenb`, and set `cs8`.

`-oddp` disable `parenb` and `parodd`, and set `cs8`.

`-spacep` disable `parenb` and `parext`, and set `cs8`.

`-markp` disable `parenb`, `parodd`, and `parext`, and set `cs8`.

`raw` (`-raw` or `cooked`)
enable (disable) raw input and output (no `ERASE`, `KILL`, `INTR`, `QUIT`, `SWTCH`, `EOT`, or output post processing).

`nl` (`-nl`) unset (set) `icrnl`, `onlcr`. In addition `-nl` unsets `inlcr`, `igncr`, `ocrnl`, and `onlret`.

`lcase` (`-lcase`)
set (unset) `xcase`, `iuclc`, and `olcuc`.

`LCASE` (`-LCASE`)
same as `lcase` (`-lcase`).

`tabs` (`-tabs` or `tab3`)
preserve (expand to spaces) tabs when printing.

`ek` reset `ERASE` and `KILL` characters back to normal # and @.

`sane` resets all modes to some reasonable values.

`async` set normal asynchronous communications where clock settings are `xcibrg`, `rcibrg`, `tsetcoff` and `rsetcoff`.

Window Size

`rows` *n* set window size to *n* rows.

`columns` *n* set window size to *n* columns.

`ypixels` *n* set vertical window size to *n* pixels.

`xpixels` *n* set horizontal window size to *n* pixels.

USAGE

End-user.

SEE ALSO

termio(BA_DEV), termios(BA_OS), termiox(BA_DEV).

LEVEL

Level 1.

NAME

su – become super-user or another user

SYNOPSIS

`su [-] [`*name* `[`*arg* `...]]`

DESCRIPTION

The command `su` allows one to become another user without logging off. The default user *name* is `root` (that is, super-user). Note that if the Enhanced Security Extension is implemented, special privilege is not associated with UID 0 (`root`); becoming UID 0 only gives you access to files owned by `root`, it does not impart the ability to override system restrictions (notably, mandatory access control).

To use `su`, the appropriate password must be supplied (unless one has the appropriate privilege). If the password is correct, `su` will execute a new environment with the real and effective user and group IDs and supplementary group list set to that of the specified user. The new command interpreter will be the optional program named in the specified user's password file entry, or the default if none is specified. Normal user ID privileges are restored when the command interpreter exits.

Any additional arguments given on the command line are passed to the command interpreter.

The following statements are true only if the command interpreter named in the specified user's password file entry is `sh` [see `sh`(BU_CMD)]. If the first argument to `su` is a `-`, the environment will be changed to what would be expected if the user actually logged in as the specified user. Otherwise, the environment is passed along with the possible exception of `PATH`.

All attempts to become another user using `su` are logged.

FILES

`/etc/security/ia/master`	system's Identification and Authentication (I&A) data file
`/etc/passwd`	system's password file
`/etc/profile`	system's profile
`$HOME/.profile`	user's profile

SEE ALSO

sh(BU_CMD).

LEVEL

Level 1.

NAME

tabs – set tabs on a terminal

SYNOPSIS

`tabs` [*tabspec*] [`+m`*n*] [`-T`*type*]

DESCRIPTION

The command `tabs` sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings.

Three types of tab specification are accepted for *tabspec*: canned, repetitive, and arbitrary. If no *tabspec* is given, the default value is `-8`, *i.e.*, standard tabs. The lowest column number is 1. Note that for `tabs`, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0.

-code Gives the name of one of a set of canned tabs. The legal codes and their meanings are as follows:

`-a` 1,10,16,36,72
Assembler, IBM System/370, first format

`-a2` 1,10,16,40,72
Assembler, IBM System/370, second format

`-c` 1,8,12,16,20,55
COBOL, normal format

`-c2` 1,6,10,14,49
COBOL compact format (columns 1-6 omitted).

`-c3` 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than `-c2`. This is the recommended format for COBOL.

`-f` 1,7,11,15,19,23
FORTRAN

`-p` 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

`-s` 1,10,55
SNOBOL

`-u` 1,12,20,44
UNIVAC 1100 Assembler

In addition to these canned formats, three other types exist:

-n A repetitive specification requests tabs at columns 1+*n*, 1+2***n*, *etc.* Of particular importance is the value `-8`: this represents the standard tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value `-0`, implying no tabs at all.

n1,n2,...
The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

`-T`*type*
: The command `tabs` usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. The argument *type* is a terminal name. If no `-T` flag is supplied, `tabs` searches for the environment variable `TERM`. If no *type* can be found, `tabs` tries a sequence that will work for many terminals.

`+m`*n*
: The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n*`+1` the left margin. If `+m` is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by `+m0`. The margin for most terminals is reset only when the `+m` flag is given explicitly.

Tab and margin setting is performed via the standard output.

USAGE

End-user.

LEVEL

Level 1.

NAME

tar – file archiver

SYNOPSIS

`tar` [*options*] [*file* ...]

DESCRIPTION

The command `tar` creates archives of files; it is often used to save files on (and restore from) magnetic tape. Its actions are controlled by the *option* argument. The *option* is a string of characters containing at most one function letter and possibly one or more modifiers. Other arguments to the command are file (or directory) names specifying which files are to be archived or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the option is specified by one of the following letters:

`r` The named files are written on the end of the archive.

`x` The named files are extracted from the archive. If a named *file* matches a directory whose contents had been written onto the archive, this directory is (recursively) extracted. If a named *file* in the archive does not exist on the system, the file is created with the same mode as the one in the archive, except that the set-user-ID and set-group-ID modes are not set unless the user has appropriate privileges. If the files exist, their modes are not changed except as described above. The owner, group, and modification time are restored (if possible). If no *file* argument is given, the entire content of the archive is extracted. Note that if several files with the same name are in the archive, the last one overwrites all earlier ones.

`t` The names of all the files in the archive are listed.

`u` The named files are added to the archive if they are not already there, or have been modified since last written into the archive. This option implies option `r`.

`c` Create a new archive; writing begins at the beginning of the archive, instead of after the last file. This option implies the `r` option.

The following characters may be used in addition to the letter that selects the desired function:

`v` Normally, `tar` does its work silently. The `v` (verbose) modifier causes it to type the name of each file it treats, preceded by the option letter. With the `t` option, `v` gives more information about the archive entries than just the name.

`w` Causes `tar` to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with `y` is given, the action is performed. Any other input means the action is not performed (`no`). This modifier is invalid with the `t` option.

`f` Causes `tar` to use the next argument as the name of the archive instead of the default, which is usually a tape drive. If the name of the *file* is `–`, `tar` writes to the standard output or reads from the standard input, whichever is appropriate. Thus, `tar` can be used as the head or tail of a pipeline. The

command `tar` can also be used to move directory hierarchies with the command:

```
(cd fromdir; tar cf - .) | (cd todir; tar xf -)
```

`b` Causes `tar` to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This option should only be used with (raw) magnetic tape archives (see `f` above). The block size is determined automatically when reading tapes (options `x` and `t`).

`l` Tells `tar` to report if it cannot resolve all of the links to the files being archived. If `l` is not specified, no error messages are printed. This modifier is valid only with the options `c`, `r`, and `u`.

`m` Tells `tar` not to restore the modification times. The modification time of the file will be the time of extraction. This modifier is invalid with the `t` option.

`o` Causes extracted files to take on the user and group identifier of the user running the program rather than those on the archive. This modifier is valid only with the `x` option.

`L` Follow symbolic links. This modifier causes symbolic links to be followed. By default, symbolic links are not followed. This modifier is valid only with the c, r, and u options.

The data interchange format used by `tar` conforms with that specified by the POSIX 1003.1-1988 standard.

ERRORS

The command `tar` reports bad option characters and read / write errors.
It also reports an error if enough memory is not available to hold the link tables.

FUTURE DIRECTIONS

The functionality of `tar` will be eventually provided by `cpio` [see cpio(BU_CMD)]. Therefore, support for `tar` will be discontinued in the future.

LEVEL

Level 1.

NAME

tty – get the name of the terminal

SYNOPSIS

`tty [-s]`

DESCRIPTION

The command `tty` prints the pathname of the user's terminal. The `-s` option suppresses printing of the terminal pathname, allowing one to test just the exit code.

ERRORS

Exit codes:

0 if standard input is a terminal,

1 otherwise.

2 if invalid options were specified,

An error is reported if the standard input is not a terminal and `-s` is not specified.

LEVEL

Level 1.

NAME

uucp, uulog, uuname – system-to-system copy

SYNOPSIS

`uucp` [`-c`][`-C`][`-d`][`-f`][`-j`][`-m`][`-n`*user*][`-r`] *source-files destination-file*

`uulog` [`-s`*system*]

`uuname` [`-l`]

DESCRIPTION

uucp

The command `uucp` copies files named by the *source-file* arguments to the *destination-file* argument. A source-filename may be a pathname on your machine, or may have the form:

system-name!*pathname*

where *system-name* is taken from a list of system names that `uucp` knows about. The destination *system-name* may also be a list of names such as

system-name!*system-name*!...!*system-name*!*pathname*

in which case, an attempt is made to send the file via the specified route to the destination. Care should be taken to ensure that intermediate nodes in the route are willing to forward information.

The shell metacharacters `?`, `*`, and `[...]` appearing in *pathname* will be expanded on the appropriate system.

Pathnames may be one of:

(1) a full pathname.

(2) a pathname preceded by `~`*name* where *name* is a login name on the specified system and is replaced by that user's login directory. Note that if an invalid login is specified, the default will be to the public directory (`$PUBDIR`).

(3) a pathname specified as `~/`*dest*, where the destination *dest* is appended to `$PUBDIR`. Note that this destination will be treated as a filename unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example, `~/dan/` as the destination will make the directory `$PUBDIR/dan` if it does not exist and put the requested file(s) in that directory.

(4) anything else is prefixed by the current directory.

If the result is an erroneous pathname for the remote system, the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

The command `uucp` sets the appropriate read and write permissions and removes execute permissions across the transmission.

The following options are interpreted by `uucp`:

`-c` Do not copy local file to the spool directory for transfer to the remote machine (default).

`-C` Force the copy of local files to the spool directory for transfer.

`-d` Make all necessary directories for the file copy (default).

`-f` Do not make intermediate directories for the file copy.

`-j` Output the job identification string on the standard output. This job identification can be used by `uustat` to obtain the status or terminate a job.

`-m` Send mail to the requester when the copy is completed.

`-n`*user*
Notify *user* on the remote system that a file was sent.

`-r` Do not start the file transfer; just queue the job.

`uulog`
The command `uulog` queries a log file of `uucp` or `uux` transactions.

If the `-s` option is specified, then `uulog` prints information about file transfer work involving system `system`.

`uuname`
The command `uuname` lists the `uucp` names of known systems. The `-l` option returns the local system name.

USAGE

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted.

SEE ALSO

mail(BU_CMD), uustat(AU_CMD), uux(AU_CMD).

LEVEL

Level 1.

NAME

uuencode, uudecode – encode a binary file, or decode its ASCII representation

SYNOPSIS

uuencode [*source-file*] *file-label*

uudecode [*encoded-file*]

DESCRIPTION

uuencode converts a binary file into an ASCII-encoded representation that can be sent using mail(BU_CMD). It encodes the contents of *source-file*, or the standard input if no *source-file* argument is given. The *file-label* argument is required. It is included in the encoded file's header as the name of the file into which uudecode is to place the binary (decoded) data. uuencode also includes the ownership and permission modes of *source-file*, so that *file-label* is recreated with those same ownership and permission modes.

uudecode reads an *encoded-file*, strips off any leading and trailing lines added by mailer programs, and recreates the original binary data with the filename and the mode and owner specified in the header.

The encoded file is an ordinary ASCII text file and can be edited by any text editor. But it is best only to change the mode or file-label in the header to avoid corrupting the decoded binary.

USAGE

Application Program.

The encoded file's size is expanded by 35% (3 bytes become 4, plus control information), causing it to take longer to transmit than the equivalent binary.

The user on the remote system who is invoking uudecode (typically uucp) must have write permission on the file specified in the *file-label*.

Because both uuencode and uudecode run with user ID set to uucp, uudecode can fail with permission denied when attempted in a directory that does not have write permission allowed for other.

SEE ALSO

mail(BU_CMD), uucp(AU_CMD), uux(AU_CMD).

LEVEL

Level 1.

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

```
uustat [-m]
uustat [-q]
uustat [-kjobid]
uustat [-rjobid]
uustat [-ssys] [-uuser]
```

DESCRIPTION

The command `uustat` will display the status of, or cancel, previously specified `uucp` commands, or provide general status on `uucp` connections to other systems.

When no options are given, `uustat` outputs the status of all `uucp` requests issued by the current user.

Not all combinations of options are valid. Only one of the following options can be specified with `uustat`:

`-m` List the status for all machines.

`-q` List the status for each machine for which there are jobs queued.

`-k`*jobid*
Kill the `uucp` request whose job identification is *jobid*. The killed `uucp` request must belong to the person issuing the `uustat` command unless that user is the super-user.

`-r`*jobid*
Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the job's modification time reaches the limit imposed by the daemon.

The options below may not be used with the ones listed above; however, these options may be used singly or together:

`-s` *sys*
Report the status of all `uucp` requests for remote system *sys*.

`-u` *user*
Report the status of all `uucp` requests issued by *user*.

SEE ALSO

uucp(AU_CMD).

LEVEL

Level 1.

NAME

uuto, uupick – public system-to-system file copy

SYNOPSIS

`uuto` [`-p`] [`-m`] *source-files destination*

`uupick` [`-s`*system*]

DESCRIPTION

uuto

The command `uuto` sends *source-files* to *destination*. The command `uuto` uses the `uucp` facility to send files, while it allows the local system to control the file access. A *source-file* name is a pathname on the user's machine. Destination has the form: *system*!...!*system*!*user* where *system* is taken from a list of system names that `uucp` knows about [see uuname in uucp(AU_CMD).] The argument *user* is the login name of someone on the specified system.

Two *options* are available:

`-p` Copy the source file into the spool directory before transmission.

`-m` Send mail to the sender when the copy is complete.

The files (or subtrees if directories are specified) are sent to a public directory (`$PUBDIR`) on *system*. Specifically, the files are sent to the directory

`$PUBDIR/receive/`*user*`/`*fsystem*,

where *user* is the recipient, and *fsystem* is the sending system.

The recipient is notified by `mail` of the arrival of files.

uupick

The command `uupick` may be used by a user to accept or reject the files transmitted to the user. Specifically, `uupick` searches `$PUBDIR` on the user's system for files sent to the user. For each entry (file or directory) found, one of the following messages is printed on the standard output:

`from` *system*`: dir` *dirname* `?`
`from` *system*`: file` *filename* `?`

The command `uupick` then reads a line from the standard input to determine the disposition of the file. The user's possible responses are:

<newline> Go on to next entry.

`d` Delete the entry.

`m` [*dir*] Move the entry to named directory *dir*. If *dir* is not specified as a complete pathname, a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.

`a` [*dir*] Same as `m` except moving all the files sent from *system*.

`p` Print the content of the file to standard output.

`q` Stop and exit.

EOT (CTRL-D.) Same as q.

!*command* Escape to the command interpreter to execute *command*.

* Print a usage summary for uuto.

The command uupick invoked with the -s*system* option will only search for files (and list any found) sent from *system*.

SEE ALSO

mail(BU_CMD), uucp(AU_CMD), uustat(AU_CMD), uux(AU_CMD).

LEVEL

Level 1.

NAME

uux – remote command execution

SYNOPSIS

uux [*options*] *command-string*

DESCRIPTION

The command uux will gather zero or more files from various systems, execute a command on a specified system, and then send the standard output of the command to a file on a specified system.

The *command-string* is made up of one or more arguments that are similar to normal command arguments, except that the command and any filenames may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

The following statements are relevant if sh is the command interpreter.

The metacharacter * will not give the desired result.

The redirection tokens >> and << are not implemented.

A filename may be specified as for uucp: it may be a full pathname, a pathname preceded by ~*name* (which is replaced by the corresponding login directory), a pathname specified as ~/*dest* (*dest* is prefixed by $PUBDIR), or a simple filename (which is prefixed by the current directory) [see uucp(AU_CMD)].

For example, the command:

```
uux "!diff a!/usr/dan/file1 b!/a4/dan/file2 > !~/dan/diff.out"
```

will get the *file1* and *file2* files from the usg and pwba machines, execute diff, and put the results in file.diff in the local PUBDIR/dan directory. (PUBDIR is the uucp public directory on the local system.)

The execution of commands on remote systems takes place in an execution directory known to the uucp system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the non-local filenames (those without path or machine reference) must be unique within the uux request. The following command will not work:

```
uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

because the file xyz will be copied from the b system as well as the c system, causing a name conflict. The command:

```
uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

will work (provided diff is a permitted command), because the local file xyz (which is not copied) does not conflict with the copied file xyz from the c system.

Any characters special to the command interpreter should be quoted either by quoting the entire *command-string* or quoting the special characters as individual arguments.

The command uux will attempt to get all files to the execution system. For files that are output files, the filename must be escaped using parentheses. For example, the command:

```
uux a!cut -f1 b!/usr/file \(c!/usr/file\)
```

gets /usr/*file* from system b, sends it to system a, performs a cut command on that file, and sends the result of the cut command to system c.

The command uux will notify the user (by mail) if the requested command on the remote system was disallowed. This notification can be turned off by the -n option. The response comes by mail from the remote machine.

The following *options* are interpreted by uux:

- \- The standard input to uux is made the standard input to the *command-string*.
- -c Do not copy local file(s) to the spool directory for transfer to the remote machine (default).
- -C Force the copy of local file(s) to the spool directory for transfer.
- -j Output the job identification string on the standard output. This job identification can be used by uustat to obtain the status or terminate a job.
- -n Do not notify the user if the command fails.
- -r Do not start the file transfer, just queue the job.
- -z Send success notification to the user.

USAGE

Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from uux. Many sites will permit little more than the receipt of mail via uux.

Only the first command of a pipeline [see sh(BU_CMD)] may have a *system-name*!. All other commands are executed on the system of the first command.

SEE ALSO

mail(BU_CMD), sh(BU_CMD), uucp(AU_CMD), uustat(AU_CMD).

LEVEL

Level 1.

NAME

vi – screen-oriented (visual) display editor

SYNOPSIS

`vi` [`-r`*file*] [`-l`] [`-w`*n*] [`-R`] [`+`*command*] *file* ...

DESCRIPTION

`vi` (visual) is a display-oriented text editor. It is based on the underlying line editor ex(AU_CMD); it is possible to switch back and forth between the two, and to execute `ex` commands from within `vi`.

When using `vi`, the terminal screen acts as a window into the file being edited. Changes made to the file are reflected in the screen display; the position of the cursor on the screen indicates the position within the file.

The environment variable `TERM` must give the terminal type; the terminal must be defined in the *terminfo* database. As with `ex`, editor initialization scripts can be placed in the environment variable `EXINIT`, or the file `.exrc` in the current or home directory.

Options

The following options are interpreted by `vi`:

`-r`*file* Recover *file* after an editor or system crash. If *file* is not specified, a list of all saved files will be printed.

`-l` set `LISP` mode (see **Edit Options** below).

`-w`*n* Set the default window size to *n* lines.

`-R` Read-only mode; the `readonly` flag is set, preventing accidental overwriting of the file.

`+`*command*
The specified `ex` command is interpreted before editing begins.

VI Commands — General Remarks

See ex(AU_CMD) for the complete description of `ex`. Only the `visual` mode of the editor is described here.

When invoked, `vi` is in *command* mode; *input* mode is entered by typing any of several commands used to insert or change text. When in input mode, `ESC` (escape) is used to leave input mode; in command mode, it is used to cancel a partial command. The terminal bell is sounded if the editor is not in input mode and there is no partially entered command.

The last (bottom) line of the screen is used to echo the input for search commands (`/` and `?`), `ex` commands (`:`), and system commands (`!`). It is also used to report errors or print other messages.

An interrupt (`BREAK` or `DEL`) typed during text input, or during the input of a command on the bottom line, terminates the input (or cancels the command) and returns the editor to command mode. During command mode an interrupt causes the bell to be sounded; in general the bell indicates an error (such as unrecognized key).

Lines displayed on the screen containing only a ~ indicate that the last line above them is the last line of the file (the ~ lines are past the end of the file). On a terminal with limited local intelligence, there may be lines on the screen marked with an @: these indicate space on the screen not corresponding to lines in the file. (These lines may be removed by entering a CTRL-R, forcing the editor to retype the screen without these holes.)

vi can process and display characters from supplementary character sets using a consistent user interface.

All processing is in character units, not columns or bytes. Accordingly, in *command mode*, vi recognizes arguments to indicate the number of characters.

In regular expressions, also, processing is performed on characters, not bytes.

If a multi-column character is split over two lines, the columns that are forced onto a new line are prepended by the same number of ASCII > characters as the original multi-column character's column width.

Command Summary

Most commands accept a preceding number as an argument, either to give a size or position (for display or movement commands), or as a repeat count (for commands that change text). For simplicity, this optional argument will be referred to as count when its effect is described.

The following operators can be followed by a movement command to specify an extent of text to be affected: c, d, y, <, >, !, and =. The region specified is from the current cursor position to just before the cursor position indicated by the move. If the command operates on lines only, then all the lines which fall partly or wholly within this region are affected. Otherwise the exact marked region is affected.

In the following, control characters are indicated in the form ^X, which stands for CTRL-X. The intended ASCII character name is also given.

Unless otherwise specified, the commands are interpreted in command mode and have no special effect in input mode.

^B (STX) Scrolls backward to display the window above the current one. A count specifies the number of windows to go back. Two lines of overlap are kept if possible.

^D (EOT) Scrolls forward a half-window of text. A count gives the number of (logical) lines to scroll, and is remembered for future ^D and ^U commands.

In input mode, backs shiftwidth spaces over the indentation provided by autoindent or ^T.

^E (ENQ) Scrolls forward one line, leaving the cursor where it is if possible.

^F (ACK

^H (BS) Moves one space to the left (stops at the left margin). A count specifies the number of spaces to back up. (Same as h.)

In input mode, backs over the last input character without erasing it.

^J (LF) Moves the cursor down one line in the same column. A count specifies the number of lines to move down. (Same as ^N and j.)

^L (FF) Clears and redraws the screen. (Used when the screen becomes scrambled for any reason.)

^M (CR) Moves to the first non-white character in the next line. A count specifies the number of lines to go forward.

^N (SO) Same as ^J and j.

^P (DLE) Moves the cursor up one line in the same column. A count specifies the number of lines to move up. (Same as k.)

^R (DC2) Redraws the current screen, eliminating the false lines marked with '@' (which do not correspond to actual lines in the file).

^T (DC4) In input mode, if at the beginning of the line or preceded only by white space, inserts shiftwidth white space. This inserted space can only be backed over using ^D.

^U (NAK) Scrolls up a half-window of text. A count gives the number of (logical) lines to scroll, and is remembered for future ^D and ^U commands.

^V (SYN) In input mode, quotes the next character to make it possible to insert special characters (including ESC) into the file.

^W (ETB) In input mode, backs up one word; the deleted characters remain on the display.

^Y (EM) Scrolls backward one line, leaving the cursor where it is if possible.

^[(ESC) Cancels a partially formed command; sounds the bell if there is none.

In input mode, terminates input mode.

When entering a command on the bottom line of the screen (ex command line or search pattern with / or ?), terminates input and executes command.

<space> Moves one space to the right (stops at the end of the line). A count specifies the number of spaces to go forward. (Same as l.)

! An operator which passes specified lines from the buffer as standard input to the specified system command, and replaces those lines with the standard output from the command. The ! is followed by a movement command specifying the lines to be passed (lines from the current position to the end of the movement) and then the command (terminated as usual by a return). A count preceding the ! is passed on to the movement command after !.

Doubling ! and preceding it by a count causes that many lines, starting with the current line, to be passed.

`"` Precedes a buffer specification. There are numbered buffers `1-9` in which the editor places deleted text. The named buffers ASCII `a-z` are available to the user for saving deleted or yanked text.

`$` Moves to the end of the current line. A count specifies the number of lines to go forward. (*e.g.*, `2$` goes to the end of the next line.)

`%` Moves to the parenthesis or curly brace which matches the parenthesis or brace at the current cursor position.

`&` Same as the `ex` command `&` (repeats previous substitute command).

`´` Single quote. When followed by a ´, returns to the previous context, placing the cursor at the beginning of the line. (The previous context is set whenever a non-relative move is made.) When followed by an ASCII lower case letter `a-z`, returns to the line marked with that letter (see the `m` command), at the first non-white character in the line.

When used with an operator such as `d` to specify an extent of text, the operation takes place over complete lines. (See also `` ` ``.)

`` ` `` When followed by a `` ` ``, returns to the previous context, placing the cursor at the character position marked. (The previous context is set whenever a non-relative move is made.) When followed by an ASCII lower case letter `a-z`, returns to the line marked with that letter (see the `m` command), at the character position marked.

When used with an operator such as `d` to specify an extent of text, the operation takes place from the exact marked place to the current position within the line. (See also ´.)

`[[` Backs up to the previous section boundary. A section is defined by the value of the `sections` option. Lines which start with a formfeed (`^L` character) or `{` also stop `[[`.

If the option *lisp* is set, stops at each `(` at the beginning of a line.

`]]` Moves forward to a section boundary (see `[[`).

`^` Moves to the first non-white position on the current line.

`(` Moves backward to the beginning of a sentence. A sentence ends at a `.` `!` or `?` which is followed by either the end of a line or by two spaces. Any number of closing `)` `]` `"` and `` ` `` characters may appear between the `.` `!` or `?` and the spaces or end of line. A count moves back that many sentences.

If the *lisp* option is set, moves to the beginning of a LISP s-expression. Sentences also begin at paragraph and section boundaries (see `{` and `[[`).

`)` Moves forward to the beginning of a sentence. A count moves forward

`}` Moves forward to the beginning of the next paragraph. A count specifies the number of paragraphs to move forward. (See `{`.)

`|` Requires a count; the cursor is placed in that column (if possible).

`+` Moves to the first non-white character in the next line. A count specifies the number of lines to go forward. (Same as `^M`.)

`,` Comma. Reverse of the last `f`, `F`, `t` or `T` command, looking the other way in the current line. A count is equivalent to repeating the search that many times.

`-` Moves to the first non-white character in the previous line. A count specifies how many lines to move back.

`.` Repeats the last command which changed the buffer. A count is passed on to the command being repeated.

`/` Reads a string to be interpreted as a regular expression. The `/` and expression are echoed on the bottom line as they are read. The search begins when return is entered to terminate the pattern. Scans forward for the next occurrence of a matching string. The search may be terminated with an interrupt (or `DEL`).

When used with an operator to specify an extent of text, the defined region is from the current cursor position to the beginning of the matched string. Whole lines may be specified by giving an offset from the matched line (using a closing / followed by a +n or -n).

Regular expressions are described in ex(AU_CMD).

`0` Moves to the first character on the current line. (Is not interpreted as a command when preceded by a non-zero digit.)

`:` Begins an `ex` command. The `:`, as well as the entered command, is echoed on the bottom line; it is executed when the input is terminated by entering a return.

`;` Repeats the last `f`, `F`, `t` or `T` command. A count is equivalent to repeating the search that many times.

`<` Shifts lines left one `shiftwidth`. May be followed by a move to specify lines. A count is passed through to the move command.

When repeated (`<<`), shifts the current line (or count lines starting at the current one).

`~` Changes the case of the current letter from upper to lower or from lower to upper. A preceding count affects that many characters. If the current character is not a letter, `~` has no effect.

`>` Shifts lines right one `shiftwidth`. (See `<`.)

`=` If the `lisp` option is set, then reindents the specified lines, as though they were typed in with *lisp* and *autoindent* set. May be preceded by a count to indicate how many lines to process, or followed by a move command for the same purpose.

`?` Scans backwards, the reverse of `/`. (See `/`.)

`A` Appends at the end of line. (Same as `$a`.)

`B` Backs up a word, where a word is any non-blank sequence, placing the cursor at the beginning of the word. A count gives the number of words to go back.

`C` Changes the rest of the text on the current line. (Same as `c$`.)

`D` Deletes the rest of the text on the current line. (same as `d$`.)

`E` Moves forward to the end of a word, where a word is any non-blank sequence. A count gives the number of words to go forward.

`F` Must be followed by a single character; scans backwards in the current line for that character, moving the cursor to it if found. A count is equivalent to repeating the search that many times.

`G` Goes to the line number given as preceding argument, or the end of the file if no preceding count is given.

`H` Moves the cursor to the top line on the screen. If a count is given, then the cursor is moved to that line on the screen, counting from the top. The cursor is placed on the first non-white character on the line. If used as the target of an operator, full lines are affected.

`I` Inserts at the beginning of a line. (Same as `^` followed by `i`.)

`J` Joins the current line with the next one, supplying appropriate white space: one space between words, two spaces after a period, and no spaces at all if the first character of the next line is `)`. A count causes that many lines to be joined rather than two.

`L` Moves the cursor to the first non-white character of the last line on the screen. A count moves to that line counting from the bottom. When used with an operator, whole lines are affected.

`M` Moves the cursor to the middle line on the screen, at the first non-white position on the line.

`N` Scans for the next match of the last pattern given to `/` or `?`, but in the reverse direction; this is the reverse of `n`.

`O` Opens a new line above the current line and enters input mode.

`P` Puts the last deleted text back before/above the cursor. The text goes back as whole lines above the cursor if it was deleted as whole lines. Otherwise the text is inserted just before the cursor.

May be preceded by a named or numbered buffer specification (`"`*x*), to retrieve the contents of the buffer.

`Q` Quits from *vi* and enters *ex* command mode.

`R` Replaces characters on the screen with characters entered, until the input is terminated with `ESC`.

`S` Changes whole lines (same as `cc`). A count changes that many lines.

`T` Must be followed by a single character; scans backwards in the current line for that character, and if found, places the cursor just after that character. A count is equivalent to repeating the search that many times.

`U` Restores the current line to its state before the cursor was last moved to it.

`W` Moves forward to the beginning of a word in the current line, where a word is a sequence of non-blank characters. A count specifies the number of words to move forward.

`X` Deletes the character before the cursor. A count repeats the effect, but only characters on the current line are deleted.

`Y` Places (yanks) a copy of the current line into the unnamed buffer (same as `yy`). A count copies that many lines. May be preceded by a buffer name to put the copied line(s) in that buffer.

`ZZ` Exits the editor, writing out the buffer if it was changed since the last write. (Same as the `ex` command `x`.)

`a` Enters input mode, appending the entered text after the current cursor position. A count causes the inserted text to be replicated that many times, but only if the inserted text is all on one line.

`b` Backs up to the beginning of a word in the current line. A word is a sequence of alphanumerics, or a sequence of special characters. A count repeats the effect.

`c` Deletes the specified region of text, and enters input mode to replace it with the entered text. If more than part of a single line is affected, the deleted text is saved in the numeric buffers. If only part of the current line is affected, then the last character to be deleted is marked with a `$`. A count is passed through to the move command. When followed by `c`, deletes current line and enters input mode.

`d` Deletes the specified region of text. If more than part of a line is affected, the text is saved in the numeric buffers. A count is passed through to the move command. When followed by `d`, deletes current line.

`e` Moves forward to the end of the next word, defined as for `b`. A count repeats the effect.

`f` Must be followed by a single character; scans the rest of the current line for that character, and moves the cursor to it if found. A count repeats the find that many times.

`h` Moves the cursor one character to the left. (Same as `^H`.) A count repeats the effect.

`i` Enters input mode, inserting the entered text before the cursor. (See `a`.)

`j` Moves the cursor one line down in the same column. (Same as `^J` and `^N`.)

`k` Moves the cursor one line up. (Same as `^P`.)

`l` Moves the cursor one character to the right (same as <space>).

`m` Must be followed by a single ASCII lower case letter *x*; marks the current position of the cursor with that letter. The exact position is referred to by `` `x ``; the line is referred to by `´x`.

`n` Repeats the last `/` or `?` scanning commands.

`o` Opens a line below the current line and enters input mode; otherwise like `O`.

`p` Puts text after/below the cursor; otherwise like `P`.

`r` Must be followed by a single character; the character under the cursor is replaced by the specified one. (The new character may be a newline.) A count replaces each of the following count characters with the single character given.

`s` Deletes the single character under the cursor, and enters input mode; the entered text replaces the deleted character. A count specifies how many characters from the current line are changed. The last character to be changed is marked with a $, as for `c`.

`t` Must be followed by a single character; scans the rest of the line for that character. The cursor is moved to just before the character, if it is found. A count is equivalent to repeating the search that many times.

`u` Reverses the last change made to the current buffer. If repeated, will alternate between these two states, thus is its own inverse. When used after an insert which inserted text on more than one line, the lines are saved in the numeric buffers.

`w` Moves forward to the beginning of the next word, where word is the same as in `b`. A count specifies how many words to go forward.

`x` Deletes the single character under the cursor. With a count, deletes that many characters forward from the cursor position, but only on the current line.

`y` Must be followed by a movement command; the specified text is copied (yanked) into the unnamed temporary buffer. If preceded by a buffer specification, `"`*x*, the text is placed in that buffer also. When followed by `y`, same as `Y`.

`z` Redraws the screen with the current line placed as specified by the following character; <return> specifies the top of the screen, `.` the center of the screen, and `-` the bottom of the screen. A count may be given after the z and before the following character to specify the new screen size for the redraw. A count before the z gives the number of the line to place in the center of the screen instead of the default current line.

SEE ALSO

ex(AU_CMD).

FUTURE DIRECTIONS

To conform to the command syntax standard, the *+command* option will be changed to the form *-c command*. The old form will continue to be accepted for some time.

LEVEL

Level 1.

The *+command* option is Level 2, effective September 30, 1989.

NAME

wall – write to all users

SYNOPSIS

`wall`

DESCRIPTION

The command `wall` reads its standard input until an end-of-file is received. It then prints this message on the terminals of all users currently logged in, preceded by:

Broadcast Message from *login-id*

The `wall` command must be invoked with appropriate privileges to override any protections the other users may have invoked [see mesg(AU_CMD)].

The `wall` command uses the locale of the sender to determine printability. `wall` will detect non–printable characters before sending them to the user's terminal. Control characters will appear as a '^' followed by the appropriate ASCII character; non-printable characters with the high-order bit set will appear in meta notation. For example, '`\003` ' is displayed as '`^C`' and '`\372`' as '`M-z`'.

USAGE

Administrator.

The command `wall` is used to warn all users, typically prior to shutting down the system.

SEE ALSO

mesg(AU_CMD), write(AU_CMD).

LEVEL

Level 1.

NAME

who – who is on the system

SYNOPSIS

`who [-uTlHqpdbrtas] [`*file*`]`

`-b` This option indicates the time and date of the last reboot.

`-r` This option indicates the current run level of the `init` process.

`-t` This option indicates the last change to the system clock.

`-a` This option turns on all options except the `-q`, `-s`, and `-H` options.

`-s` This option is the default and lists only the *name, line,* and *time* fields.

SEE ALSO

getut(SD_LIB).

LEVEL

Level 1.

NAME

write – write to another user

SYNOPSIS

`write` *user* [*terminal*]

DESCRIPTION

The command `write` copies lines from the user's terminal to that of another user. When first called, it sends the message:

`Message from` *sender-login-id* `(`*terminal)* [*date*] ...

to the user addressed. When it has successfully completed the connection, it also sends two bells to the sender's terminal to indicate that what the sender is typing is being sent.

The recipient of the message should write back, by typing `write` *sender-login-id*, on receipt of the initial message. Whatever each user types (except for command escapes, see below) is printed on the other user's terminal, until an end-of-file or an interrupt is sent. At that point `write` writes `EOT` on the other terminal and exits. The recipient can also stop further messages from coming in by executing `mesg n`.

To write to a user who is logged in more than once, the *terminal* argument may be used to indicate which terminal to send to (e.g., `term/12`); otherwise, the first writable instance of the user found in an implementation-defined database is assumed and an informational message is written.

A user may deny or grant write permission by use of the `mesg` command. Certain commands disallow messages in order to prevent interference with their output. However, if the sender has appropriate privileges, messages can be forced onto a write-inhibited terminal.

If the character `!` is found at the beginning of a line, `write` calls the command interpreter to execute the rest of the line as a command.

`write` uses the locale of the sender to determine printability. `write` will detect non-printable characters before sending them to the user's terminal. Control characters will appear as a '`^`' followed by the appropriate ASCII character; non-printable characters with the high-order bit set will appear in meta notation. For example, '`\003`' is displayed as '`^C`' and '`\372`' as '`M-z`'.

ERRORS

The following errors are reported:

- the user addressed is not logged on.
- the user addressed denies write permission [see mesg(AU_CMD)].
- the user's terminal is set to `mesg n` and the recipient cannot respond.
- the recipient changes permission (`mesg n`) after `write` had begun.

SEE ALSO

getut(SD_LIB), mesg(AU_CMD), who(AU_CMD).

LEVEL

Level 1.

Administered Systems Introduction

The Administered System Extension is composed primarily of utilities used for system administration. Many of these are, in fact, restricted to the user with appropriate privilege.

The following are prerequisite for support of the Administered Systems Extension:

- Base System
- Kernel Extension
- Basic Utilities Extension
- Advanced Utilities Extension

Summary of Commands and Utilities

The following commands and utilities are supported by the Administered Systems Extension (*exception:* items marked with a (#) are optional). Items marked with a (*) are Level 2, as defined in the *General Introduction* to this volume. Items marked with a ( ) have been internationalized (see `envvar(BA_ENV)`). Items marked with a (†) are new to this edition of the SVID.

acctcms* **fdp** **lastlogin*** **pkgparam** **setmnt***
acctcom* **ffile** **link*** **pkgproto** **setuname**
acctcon1* **fimage** **logins** **pkgrm** **shutacct***
acctcon2* **fmtmsg*** **logkeeper** **pkgtrans** **srchtxt**
acctdisk* **fsck** **migration** **prctmp*** **startup***
acctmerg* **fsdb** **mkfs** **prdaily*** **sync**
accton* **fstyp** **mkmsgs** **prtacct*** **sysdef**
acctprc1* **fuser** **mknod** **prtconf** **timex***
acctprc2* **fwtmp*** **modadmin†** **pwck** **turnacct***
acctwtmp* **groupadd** **monacct*** **removef** **umount**
backup **groupdel** **mount** **restore** **unlink***
bkexcept **groupmod** **msgalert** **rsnotify** **urestore**
bkhistory **grpck** **msglog** **rsoper** **ursstatus**
bkoper **incfile** **msgrpt** **rsstatus** **useradd**
bkreg **init** **mvdir** **runacct*** **userdel**
bkstatus **installf** **nice*** **sa1*** **usermod**
chargefee* **ipcrm** **pkgadd** **sa2*** **volcopy***
ckpacct* **ipcs** **pkgask** **sacadm** **whodo**
devnm **killall** **pkgchk** **sadc*** **wtmpfix***
diskusg* **labelit** **pkginfo** **sadp#*** **zdump**
dodisk* **last** **pkgmk** **sar*** **zic**
fdisk

Organization of Technical Information

The ‘‘Administered Systems Commands and Utilities’’ chapter provides manual page descriptions of commands and utilities supported by this extension.

Administered Systems Commands And Utilities

The following section contains the manual pages for the AS_CMD routines.

FINAL COPY
June 15, 1995
File: as_cmd.cov
svid

NAME

acct: accton, acctwtmp, chargefee, ckpacct, dodisk, lastlogin, monacct, prdaily, prtacct, shutacct, startup, turnacct – miscellaneous accounting and support commands

SYNOPSIS

`accton` [*file*]

`acctwtmp` "*reason*"

`chargefee` *login-name number*

`ckpacct` [*blocks*]

`dodisk` [*files*]

`lastlogin`

`monacct` *number*

`prdaily` [`-l`][`-c`] [*mmdd*]

`prtacct` *file* ["*heading*"]

`shutacct` ["*reason*"]

`startup`

`turnacct on`

`turnacct off`

`turnacct switch`

DESCRIPTION

The accounting software provides utilities to collect data on: process accounting, connect accounting, disk usage, command usage, summary command usage, and users' last login.

The `runnacct` and `monacct` commands [see runacct(AS_CMD)] use the utilities listed here to produce daily and monthly summary files and reports that can be printed using `prdaily`; they use a number of intermediate files and support utilities that can also be used to tailor-make new accounting systems. Many of these utilities produce or manipulate "total accounting" (`tacct`) records which can be summarized by `acctmerg` [see acctmerg(AS_CMD)] and printed using `prtacct`.

The command `accton` without parameters turns process accounting off. If *file* is given, `accton` will turn accounting on. The argument *file* must be the name of an existing file (normally `/var/adm/pacct`), to which the system appends process accounting records [see acct(KE_OS)].

The command `acctwtmp` writes a `utmp` structure to its standard output. The record contains the current time and a string of characters that describe the *reason*. A record type of `ACCOUNTING` is assigned. The argument *reason* must be a string of (11 or less) characters, numbers, `$`, or spaces. For example, the following are suggestions for use in startup and shutdown procedures, respectively:

```
acctwtmp "acctg on" >> /var/adm/wtmp
acctwtmp "acctg off" >> /var/adm/wtmp
```

The command `chargefee` is invoked to charge a *number* of units to *login-name*. A `tacct` record is written to `/var/adm/fee`, to be merged with other accounting records by `acctmerg`.

The command `ckpacct` is typically initiated via the `cron` command [see cron(AU_CMD)]. It periodically checks the size of `/var/adm/pacct`. If the size exceeds *blocks*, 1000 by default, `turnacct` will be invoked with argument `switch`. If the number of free disk blocks in the `/var` file system falls below 500, `ckpacct` will automatically turn off the collection of process accounting records via the `off` argument to `turnacct`. The accounting will be activated again on the next invocation of `ckpacct` when at least this number of blocks is restored.

The command `dodisk` is typically invoked by `cron` to perform the disk accounting functions. By default, it will do disk accounting on the special files in `/etc/vfstab`. If *files* are used, they should be the special filenames of mountable file systems; disk accounting will be done on these file systems only.

The command `lastlogin` is invoked (typically by `runacct`) to update `/var/adm/acct/sum/loginlog`, which shows the last date on which each person logged in.

The command `monacct` is typically invoked once each month. The argument *number* indicates which month or period it is. If *number* is not given, it defaults to the current month (01–12). This default is useful if `monacct` is to be executed via `cron` on the first day of each month. The command `monacct` creates summary files in `/var/adm/acct/fiscal`, restarts summary files in `/var/adm/acct/sum`, and deletes the previous days' accounting reports (see `prdaily` below).

The command `prdaily` is invoked (typically by `runacct`) to format a report of the previous day's accounting data. The report resides in `/var/adm/acct/sum/rprt`*mmdd* where *mmdd* is the month and day of the report. The current daily accounting reports may be printed by typing `prdaily`. Previous days' accounting reports can be printed by using the *mmdd* option and specifying the report date desired. The `-l` option prints a report of exceptional usage by login ID for the specified date. Previous daily reports are removed and therefore inaccessible after each invocation of `monacct`. The `-c` option prints a report of exceptional resource usage by command, and may be used on current day's accounting data only.

The command `prtacct` can be used to format and print any total accounting (`tacct`) file.

The command `shutacct` is typically invoked during a system shutdown (usually in `/sbin/shutdown`) to turn process accounting off and append a *reason* record to `/var/adm/wtmp`.

The command `startup` is typically called by the system initialization routine to turn on process accounting whenever the system is brought up.

The command `turnacct` is an interface to `accton` to turn process accounting `on` or `off`. The `switch` argument turns accounting off, moves the current `/var/adm/pacct` to the next free name in `/var/adm/pacct`*incr* (where *incr* is a number starting with `1` and incrementing by one for each additional `pacct` file), then turns accounting back on again.

FILES

`/var/adm/wtmp`	login/logoff summary
`/etc/passwd`	used for login name to user ID conversions
`/usr/lib/acct`	directory for accounting commands
`/var/adm/fee`	accumulator for fees
`/var/adm/pacct`	current file for process accounting
`/var/adm/acct/sum`	summary directory

USAGE

Administrator.

SEE ALSO

acct(KE_OS), acctcms(AS_CMD), acctcom(AS_CMD), acctcon(AS_CMD), acctmerg(AS_CMD), acctprc(AS_CMD), cron(AU_CMD), diskusg(AS_CMD), fwtmp(AS_CMD), runacct(AS_CMD).

FUTURE DIRECTIONS

There are two specific changes being made in the near future. File sizes will be reported in terms of a specific unit size, such as 1K, independent of the file system block size. It will also become unnecessary to turn Process Accounting off and then on again when switching accounting files.

LEVEL

Level 2: September 30, 1989.

NAME

acctcms – command summary from per-process accounting records

SYNOPSIS

`acctcms [-a [-p][-o]] [-c][-j][-n][-s]` *files*

DESCRIPTION

The command `acctcms` reads one or more *files*, normally in the form produced by `acct()` [see acct(KE_OS)]. It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format.

The options have the following meanings:

`-a` Print output in a user readable, rather than in the internal summary, format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, ''hog factor'', characters transferred, and blocks read and written, as in `acctcom` [see acctcom(AS_CMD)]. Output is normally sorted by total kcore-minutes.

`-c` Sort by total CPU time, rather than total kcore-minutes.

`-j` Combine all commands invoked only once under the `***other`.

`-n` Sort by number of command invocations.

`-s` Any filenames encountered hereafter are already in internal summary format.

The following options may be used only with the `-a` option.

`-p` Output a prime-time-only command summary.

`-o` Output a non-prime (offshift) time only command summary.

When `-p` and `-o` are used together, a combination prime and non-prime time report is produced. All the output summaries will be total usage except number of times executed, CPU minutes, and real minutes which will be split into prime and non-prime.

A typical sequence for performing daily command accounting and for maintaining a running total is:

```
acctcms file . . . >today
cp total previoustotal
acctcms –s today previoustotal >total
acctcms –a –s today
```

USAGE

Administrator.

SEE ALSO

acct(AS_CMD), acct(KE_OS), acctcom(AS_CMD), acctcon(AS_CMD), acctmerg(AS_CMD), acctprc(AS_CMD), fwtmp(AS_CMD), runacct(AS_CMD).

LEVEL

Level 2: September 30, 1989.

NAME

acctcom – search and print process accounting file(s)

SYNOPSIS

`acctcom` [[*options*] [*file*]] ...

DESCRIPTION

The command `acctcom` reads *file*, the standard input, or `/var/adm/pacct`, in the form produced by `acct()` [see acct(KE_OS)] and writes selected records to the standard output. Each record represents the execution of one process and shows the `COMMAND NAME`, `USER`, `TTYNAME`, `START TIME`, `END TIME`, `REAL (SEC)`, `CPU (SEC)`, `MEAN SIZE(K)`, and optionally, `F` (the `fork/exec` flag: `1` for `fork` without `exec`), `STAT` (the system exit status), `HOG FACTOR`, `KCORE MIN`, `CPU FACTOR`, `CHARS TRNSFD`, and `BLOCKS R/W` (total blocks read and written).

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

If no *files* are specified, and if the standard input is associated with a terminal or `/dev/null`, `/var/adm/pacct` is read; otherwise, the standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward, *i.e.*, in chronological order by process completion time. The *options* and arguments for this command are:

`-a` Show average statistics about the processes selected. The statistics will be printed after the output records.

`-b` Read backwards, showing latest commands first. This *option* has no effect when the standard input is read.

`-f` Print the `fork/exec` flag and system exit status columns in the output.

`-h` Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution (the "hog factor").

`-i` Print columns containing total blocks read and written.

`-k` Instead of memory size, show total kcore-minutes.

`-m` Show mean core size (the default).

`-r` Show CPU factor (user time/(system-time + user-time)).

`-t` Show separate system and user CPU times.

`-v` Exclude column headings from the output.

`-l` *line* Show only processes belonging to terminal `/dev/`*line*.

`-u` *user* Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a # which designates only those processes executed with appropriate privileges, or ? which designates only those processes associated with unknown user IDs.

`-g` *group* Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.

`-s` *time* Select processes existing at or after *time*

`-e` *time* Select processes existing at or before *time.*

`-S` *time* Select processes starting at or after *time.*

`-E` *time* Select processes ending at or before *time.* Using the same *time* for both `-S` and `-E` shows the processes that existed at *time.*

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

`-q` Do not print any output records, just print the average statistics as with the `-a` option.

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

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

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

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

`-I` *chars* Show only processes transferring more characters than the cut-off number given by *chars.*

FILES

`/etc/passwd`
`/etc/group`
`/var/adm/pacct`

USAGE

Administrator.

SEE ALSO

acct(KE_OS), acct(AS_CMD), acctcms(AS_CMD), acctcon(AS_CMD), acctmerg(AS_CMD), acctprc(AS_CMD), ed(BU_CMD), fwtmp(AS_CMD), regcmp(BA_LIB), runacct(AS_CMD).

LEVEL

Level 2: September 30, 1989.

NAME

acctcon: acctcon1, acctcon2, prctmp – connect-time accounting

SYNOPSIS

`acctcon1` [`-p`] [`-t`] [`-l` *file*] [`-o` *file*]

`acctcon2`

`prctmp`

DESCRIPTION

The command `acctcon1` converts a sequence of login/logoff records read from its standard input to a sequence of session records, one per login session. Its input should normally be redirected from `/var/adm/wtmp`. The record format is readable, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time.

The options and arguments have the following meanings:

`-p` Print input only, showing line name, login name, and time (in both numeric and date/time formats).

`-t` `acctcon1` maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The `-t` flag causes it to use the last time found in its input, instead of the current time, thus assuring reasonable and repeatable numbers for non-current files.

`-l` *file*
: *file* is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Various events during logoff each generate logoff records, so that the number of logoffs is often three to four times the number of sessions.

`-o` *file*
: *file* is filled with an overall record for the accounting period, giving starting time, ending time, and the count and type of various accounting records produced by `acctwtmp` [see `acctwtmp` in acct(AS_CMD)].

The command `acctcon2` expects as input a sequence of login session records (as produced by `acctcon1`), and converts them into total accounting records.

The command `prctmp` can be used to print the session record file as produced by `acctcon1`.

FILES

`/var/adm/wtmp`

USAGE

Administrator.

The command `wtmpfix` [see `wtmpfix` in fwtmp(AS_CMD)] can be used to correct for the confusion caused by date changes.

EXAMPLE

These commands are typically used as shown below. The file `ctmp` can be used by `acctprc1` [see `acctprc1` in acctprc(AS_CMD)]:

```
acctcon1 –t –l lineuse –o reboots </var/adm/wtmp |
     sort +1n +2 >ctmp

acctcon2 <ctmp | acctmerg >ctacct
```

SEE ALSO

acct(AS_CMD), acct(KE_OS), acctcms(AS_CMD), acctcom(AS_CMD), acctmerg(AS_CMD), acctprc(AS_CMD), fwtmp(AS_CMD), runacct(AS_CMD).

FUTURE DIRECTIONS

The commands `acctcon1` and `acctcon2` will be merged and called `acctcon` in the near future.

LEVEL

Level 2: September 30, 1989.

NAME

acctmerg – merge or add total accounting files

SYNOPSIS

`acctmerg [-a] [-i] [-p] [-t] [-u] [-v] [`*file* `...]`

DESCRIPTION

The command `acctmerg` reads its standard input and up to nine additional files, all in the total accounting (`tacct`) format or a user readable version thereof. It merges these inputs by adding records whose keys (normally user ID and name) are identical, and expects the inputs to be sorted on those keys.

The options and arguments have the following meanings:

`-a` Produce output in a user readable version of `tacct`.

`-i` Input files are in a user readable version of `tacct`.

`-p` Print input with no processing.

`-t` Produce a single record that totals all input.

`-u` Summarize by user ID, rather than user ID and name.

`-v` Produce output in verbose user readable format, using more precise notation for floating point numbers.

USAGE

Administrator.

EXAMPLE

The following sequence is useful for making "repairs" to any file kept in this format:

```
acctmerg -v <file1 >file2
(edit file2 as desired)
acctmerg -i <file2 >file1
```

SEE ALSO

acct(KE_OS), acct(AS_CMD), acctcms(AS_CMD), acctcom(AS_CMD), acctcon(AS_CMD), acctprc(AS_CMD), fwtmp(AS_CMD), runacct(AS_CMD).

LEVEL

Level 2: September 30, 1989.

NAME

acctprc, **acctprc1**, **acctprc2** – process accounting

SYNOPSIS

acctprc

acctprc1 [*ctmp***]**

acctprc2

DESCRIPTION

acctprc reads standard input, in the form described by **acct** (AS_CMD) and converts it to total accounting records [see the **tacct** record in **acct** (AS_CMD)] **acctprc** divides CPU time into prime time and non-prime time and determines mean memory size (in memory segment units). **acctprc** then summarizes the **tacct** records, according to user IDs, and adds login names corresponding to the user IDs. The summarized records are then written to standard output. **acctprc1** reads input in the form described by **acct** (AS_CMD) adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in memory segment units). If *ctmp* is given, it is expected to contain a list of login sessions sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file, just as **acctprc** does. The information in *ctmp* helps it distinguish between different login names sharing the same user ID.

From standard input, **acctprc2** reads records in the form written by **acctprc1**, summarizes them according to user ID and name, then writes the sorted summaries to the standard output as total accounting records.

Files

/etc/passwd

USAGE

The **acctprc** command is typically used as shown below:

```
acctprc < /var/adm/pacct > ptacct
```

The **acctprc1** and **acctprc2** commands are typically used as shown below:

```
acctprc1 ctmp < /var/adm/pacct | acctprc2 > ptacct
```

SEE ALSO

acct (AS_CMD), **acctcms** (AS_CMD), **acctcom** (AS_CMD), **acctcon** (AS_CMD), **acctmerg** (AS_CMD), **cron** (AU_CMD), **fwtmp** (AS_CMD), **runacct** (AS_CMD),

LEVEL

Level 2.

NOTICES

Although it is possible for **acctprc1** to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from **cron** (AU_CMD) for example. A more precise conversion can be done using the **acctwtmp** program in **acct** (AS_CMD) **acctprc** does not distinguish between users with identical user IDs.

A memory segment of the mean memory size is a unit of measure for the number of bytes in a logical memory segment on a particular processor.

NAME

backup – initiate or control a system backup session

SYNOPSIS

`backup -i [-s][-t` *table*`][-o` *orig*`][-e|n][-c` *week*`:`*day*`][-m` *user*`]`

`backup -i [-s][-t` *table*`][-o` *orig*`][-e|n][demand][-m` *user*`]`

`backup -i [-v][-t` *table*`][-o` *orig*`][-e|n][-c` *week*`:`*day*`][-m` *user*`]`

`backup -i [-v][-t` *table*`][-o` *orig*`][-e|n][demand][-m` *user*`]`

`backup -a[-t` *table*`][-o` *orig*`][-e|n][-c` *week*`:`*day*`][-m` *user*`]`

`backup -a[-t` *table*`][-o` *orig*`][-e|n][demand][-m` *user*`]`

`backup -S [-u` *user*`]`

`backup -S [-A]`

`backup -S [-j` *jobid*`]`

`backup -R [-u` *user*`]`

`backup -R [-A]`

`backup -R [-j` *jobid*`]`

`backup -C [-u` *user*`]`

`backup -C [-A]`

`backup -C [-j` *jobid*`]`

DESCRIPTION

Without options, `backup` performs all backup operations specified for the current day and week of the backup rotation in the backup table [see bkreg(AS_[

`backup -i` establishes interactive mode, which assumes that an operator is present at the terminal where the `backup` command was issued. In this mode, `bkoper` is automatically invoked at the terminal where the `backup` command was entered. The operator responds to the prompts as they arrive.

`backup -a` establishes automatic mode, which assumes that no operator is available. In this mode, any backup operation that requires operator intervention fails. Backups that can be satisfied by mounted media proceed.

Table Validations

A number of backup service databases must be consistent before the backups listed in a backup table can be performed. These consistencies can only be validated at the time `backup` is initiated. If any of them fail, `backup` will terminate. Invoking `backup -e|n` performs the validation checks in addition to displaying the set of backup operations to be performed. The validations are:

1. The backup method must be a default method or be an executable file in `/etc/bkup/method/*`.
2. The dependencies for an entry are all defined in the table. Circular dependencies (*e.g.*, entry `abc` depends on entry `def;` entry `def` depends on entry `abc`) are not allowed.

Options

`-a` Initiates all backup operations in automatic mode; does not prompt an operator to service media.

`-c` *week*`:`*day*
`demand` Selects from the backup table only those backup operations for the specified week and day of the backup rotation, instead of the current day and week of the rotation. If `demand` is specified, selects only those backup operations scheduled to be performed on demand.

`-e` Displays an estimate of the number of media required to perform each backup operation prior to starting the backup operation.

`-i` Selects interactive operation.

`-j` *jobid* Controls only the backup job identified by *jobid*. *jobid* is a backup job ID.

`-m` *user* Sends mail to the named *user* when all backup operations for the backup job are complete.

`-n` Displays the set of backup operations that would be performed if the `n` were omitted, but does not perform the backup operations. The display is ordered according to the dependencies and priorities specified in the backup table.

`-o` *orig* Initiates backup operations only on the named originating object [see bkreg(AS_CMD) for the correct form of *orig*].

`-s` Displays a dot (.) for each 100 blocks transferred to the destination device. The dots are displayed while each backup operation is progressing.

`-t` *table* Initiates backup operations described in the specified backup table, *table*, instead of the default table.

`-u` *user* Controls backup jobs started by the named *user* instead of those started by the user invoking the command. *user* is a user login ID.

`-v` While each backup operation is progressing, display the name of each file, directory, file system partition, or data partition as soon as it has been transferred to the destination device.

`-A` Controls backup jobs for all users instead of those started by the user invoking the command.

`-C` Cancels backup jobs.

`-R` Resumes suspended backup jobs.

`-S` Suspends backup jobs.

ERRORS

The exit codes for `backup` are the following:

0 = successful completion of the task

1 = one or more parameters to `backup` are invalid.

2 = an error has occurred which caused `backup` to fail to complete all portions of its task.

FILES

`/etc/bkup/bkreg.tab`	the default backup table
`/etc/bkup/method/*`	backup methods

EXAMPLE

Example 1:

```
backup -i -v -c 2:1 -m admin3
```

initiates those backups scheduled for Monday of the second week in the rotation period instead of backups for the current day and week. Performs the backup in interactive mode and displays on standard output the name of each file, directory, file system partition, or data partition as soon as it is transferred to the destination device. When all backups are completed, sends mail notification to the user with login ID `admin3`.

Example 2:

```
backup –o /usr:/dev/rdsk/c1d0s2:usr
```

initiates only those backups from the `usr` file system that is mounted on the originating device /dev/rdsk/c1d0s2 and is labeled `usr`.

Example 3:

```
backup -S
```

Suspends the backup jobs requested by the invoking user.

Example 4:

```
backup -R -j back-359
```

resumes the backup operations included in backup job ID `back-359`.

SEE ALSO

bkhistory(AS_CMD), bkoper(AS_CMD), bkreg(AS_CMD), bkstatus(AS_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.

Optional

NAME

bkexcept – change or display an exception list for incremental backups

SYNOPSIS

`bkexcept -a` *pattern* ... [`-t` *exception_list*]

`bkexcept -r` *pattern* ... [`-t` *exception_list*]

`bkexcept` [`-d` *pattern* ...] [`-t` *exception_list*]

DESCRIPTION

The command `bkexcept` displays a list of patterns describing files that are to be excluded when backup operations occur using `incfile` [see incfile(AS_CMD)]. The list is known as the "exception list."

`bkexcept -a` adds patterns to the list.

`bkexcept -r` removes patterns from the list.

Patterns

Patterns describe individual files or sets of files. Patterns must conform to pathname naming conventions specified in the *Base System Introduction*. A pattern is taken as a filename and is interpreted in the manner of `cpio` [see cpio(BU_CMD)]. A pattern can include the shell special characters `*`, `?`, and `[]`. Asterisk (`*`) and question mark (`?`) will match period (`.`) and slash(`/`). Because these are shell special characters, they must be escaped on the command line.

There are three general methods of specifying entries to an exception list. To specify all files under a particular directory, specify the directory name (and any desired subdirectories) followed by an asterisk:

```
/directory/subdirectories/*
```

To specify all instances of a filename regardless of its location, specify the filename preceded by an asterisk:

```
*/filename
```

To specify one instance of a particular file, specify the entire pathname to the file:

```
/directory/subdirectories/filename
```

If *pattern* is a dash (–), standard input is read for a list of patterns (one per line until `EOF`) to be added or deleted.

Options

`-a` *pattern* ...
: Adds *pattern* to the exception list. *pattern* is one or a list of patterns (comma-separated or blank-separated and enclosed in quotes) describing sets of files.

`-d` *pattern* ...
: Displays entries in the exception list. If *pattern* begins with a slash (/), `-d` displays all entries whose names begin with *pattern*. If *pattern* does not begin with a slash, `-d` displays all entries that include *pattern* anywhere in the entry. If *pattern* is a dash (–), input is taken from standard input.

The entries are displayed in a collating sequence of special characters, numbers, then alphabetical order.

`-t` *exception_list*
Designates the *exception_list* on which action is to be taken. If not specified, the default exception list is assumed.

`-r` *pattern ...*
Removes *pattern* from the exception list. *pattern* is one or a list of patterns (comma-separated or blank-separated and enclosed in quotes) describing sets of files. *pattern* must be an exact match of an entry in the exception list for *pattern* to be removed. Patterns that are removed are echoed to standard output.

ERRORS

The exit codes for `bkexcept` are the following:

0 = successful completion of the task

1 = one or more parameters to `bkexcept` are invalid.

2 = an error has occurred which caused `bkexcept` to fail to complete all portions of its task.

FILES

`/etc/bkup/bkexcept.tab`	default exception list
`/var/sadm/bkup/`*table*`/bkexcept.tab`	exception list for non-default backup tables

EXAMPLE

Example 1:

```
bkexcept -a /tmp/\*,/var/tmp/\*,/usr/rje/\*,\*/trash
```

adds the four sets of files to the exception list, (all files under `/tmp`, all files under `/var/tmp`, all files under `/usr/rje`, and any file on the system named `trash`).

Example 2:

```
bkexcept -d /tmp
```

displays the following patterns from those added to the exception list in example 1.

```
/tmp/*
```

Example 3:

```
bkexcept -d tmp
```

displays the following patterns from those added to the exception list in example 1.

```
/tmp/*, /var/tmp/*
```

Example 4:

```
bkexcept -r /var/tmp/*,/usr/rje/*
```

removes the two sets of files from the exception list.

SEE ALSO

backup(AS_CMD), cpio(BU_CMD), incfile(AS_CMD), sh(BU_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.

Optional

NAME

bkhistory – report on completed backup operations

SYNOPSIS

`bkhistory [-lh] [-d` *dates*`][-o` *origs*`] [-t` *tags*`][-f` *c*`]`
`bkhistory -p` *period*

DESCRIPTION

The command `bkhistory` without options reports a summary of the contents of the backup history log, `bkhist.tab`. The list of backups is sorted by tag with the most recent backup operation listed first. `backup` updates this log after each successful backup operation.

`bkhistory` may only be executed by a user with appropriate privileges.

`bkhistory -p` assigns a rotation *period* (in weeks) for the history log; all entries older than the specified number of weeks are deleted from the log. The default rotation period is one (1) week.

The options and arguments have the following meanings:

`-d` *dates* Restricts the report to backup operations performed on the specified dates. *dates* is a list of dates in `date` format [see date(BU_CMD)]. However, only *month* is required in the option argument; *day*, *hour*, *minute*, and *year*, are optional. The list of *dates* is either comma-separated or blank-separated and surrounded by quotes.

`-f` *c* Suppresses field wrap on the display and specifies an output field separator to be used. The value of *c* is the character that will appear as the field separator on the display output. For clarity of output, do not use a separator character that is likely to occur in a field. For example, do not use the colon as a field separator character if the display will contain dates that use a colon to separate hours from minutes. To use the default field separator (tab), specify the null character (`""`) for *c*.

`-h` Suppresses headers for the reports.

`-l` Displays a long form of the report. This produces an `ls -l` listing [see ls(BU_CMD)] of the files included in the backup archive (if backup tables of contents are available on-line).

`-o` *origs* Restricts the report to the specified originating objects (file systems or data partitions). *origs* is a list of originating object *onames* or *odevices* either comma-separated or blank-separated and surrounded by quotes [see bkreg(AS_CMD) for the format of *onames* and *odevices*].

`-p` *period* Sets the number of weeks of information that will be saved in the backup history table. The minimum value of *period* is 1; there is no maximum value. By default, *period* is 1.

`-t` *tags* Restricts the report to backups with the specified *tags*. *tags* is a list of tag values as specified in a backup table. The list of *tags* is either comma-separated or blank-separated and surrounded by quotes.

ERRORS

The exit codes for `bkhistory` are the following:

0 = successful completion of the task

1 = one or more parameters to `bkhistory` are invalid.

2 = an error has occurred which caused `bkhistory` to fail to complete all portions of its task.

FILES

`/etc/bkup/bkhist.tab`	the backup history log that contains information about successfully completed backup operations
`/var/sadm/bkup/`*table*`/bkreg.tab`	describes the backup policy established by the administrator
`/var/sadm/bkup/toc`	lists directories with on-line tables of contents

EXAMPLE

Example 1:

```
bkhistory –p 3
```

sets the rotation period for the history log to three weeks. Entries older than 3 weeks are deleted from the log.

Example 2:

```
bkhistory –t SpoolDai,UsrDaily,TPubsWed
```

displays a report of completed backup operations for the three tags listed.

Example 3:

```
bkhistory –l –t acctsweekly
```

Displays an `ls –l` listing of the files that were backed up for the backup operation with tag `acctsweekly`.

SEE ALSO

backup(AS_CMD), bkreg(AS_CMD), date(BU_CMD), ls(BU_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.

Optional

NAME

bkoper – interact with backup operations to service media insertion prompts

SYNOPSIS

`bkoper [-u`

```
1. back-111 usrsun /dev/dsk/c1d0s1  disk /dev/dsk/c2d1s9 usrsave
2. back-112 fs2daily /dev/dsk/c1d0s8 ctape /dev/ctape/c4d0s2 -
```

Backup headers are numbered on the basis of arrival; the oldest header has the lowest number. If the destination device does not have a media name, a dash is displayed in the header.

SEE ALSO

bkreg(AS_CMD), bkstatus(AS_CMD), getvol(AS_CMD), mailx(AU_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.

Optional

NAME

bkreg – change or display the contents of a backup table

SYNOPSIS

`bkreg -p` *period* [`-w` *cweek*] [`-t` *table*]

`bkreg -a` *tag* `-o` *orig* `-d` *ddev* `-c` *weeks*`:`*days* `-m` *method* [`-b` *moptions*]
[`-t` *table*][`-P` *prio*][`-D` *depend*]

`bkreg -a` *tag* `-o` *orig* `-d` *ddev* `-c demand -m` *method* [`-b` *moptions*]
[`-t` *table*][`-P` *prio*][`-D` *depend*]

`bkreg -e` *tag* [`-o` *orig*][`-c` *weeks*`:`*days*] [`-m` *method*][`-d` *ddev*] [`-b` *moptions*]
[`-t` *table*][`-P` *prio*][`-D` *depend*]

`bkreg -e` *tag* [`-o` *orig*][`-c demand`] [`-m` *method*][`-d` *ddev*] [`-b` *moptions*]
[`-t` *table*][`-P` *prio*][`-D` *depend*]

`bkreg -r` *tag* [`-t` *table*]

`bkreg -C` *fields* [`-hv`][`-t` *table*][`-f` *c*][`-c` *weeks*`:`*days*]

`bkreg -C` *fields* [`-hv`][`-t` *table*][`-f` *c*][`-c demand`]

`bkreg` [`-A`][`-hsv`][`-t` *table*][`-c` *weeks*`:`*days*]

`bkreg` [`-A`][`-hsv`][`-t` *table*][`-c demand`]

`bkreg` [`-O`][`-hsv`][`-t` *table*][`-c` *weeks*`:`*days*]

`bkreg` [`-O`][`-hsv`][`-t` *table*][`-c demand`]

`bkreg` [`-R`][`-hsv`][`-t` *table*][`-c` *weeks*`:`*days*]

`bkreg` [`-R`][`-hsv`][`-t` *table*][`-c demand`]

DESCRIPTION

A backup table is a file containing descriptions of backup operations to be performed on file systems or data partitions. The default backup table is `/var/sadm/bkup/default/bkreg.tab`. Other backup tables may be created.

`bkreg` may only be executed by a user with appropriate privileges.

Originating Objects

An originating object is a disk partition formatted either as a file system [see mkfs(AS_CMD)] or as a raw data partition. An originating object is described by its originating object name, its device name, and optional volume labels.

Several backup operations for different originating objects may be active concurrently by specifying priorities and dependencies. During a backup session, all higher-priority backup operations must be active, waiting or suspended before any lower-priority backup operations are started. All backup operations of a given priority may proceed concurrently unless dependencies are specified. If one backup is declared to be dependent on others, it will not be started until all of its antecedents have completed successfully.

Destination Devices

Each backup is written to a set of storage volumes inserted into a destination device. A destination device has a destination device group, a destination device name, media characteristics, and volume labels. Default characteristics for a medium may be overridden.

Backup Methods

An originating object is backed up to a destination device archive using a *method*. The method determines the amount of information backed up and the representation of that information. Different methods may be used for a given originating object on different days of the rotation.

Several default methods are provided with the backup service. Others methods may be added by a UNIX System site. [For descriptions of the default methods, see incfile(AS_CMD), ffile(AS_CMD), fdisk(AS_CMD), fimage(AS_CMD), and fdp(AS_CMD).] Each method accepts a set of options that are specific to the method.

A backup archive may be migrated to a different destination by specifying `migration` as the backup method. The device name of the originating object for a migration must have been the destination device for a previously successful backup operation. This form of backup does not re-archive the originating object. It copies an archive from one destination to another, updating the backup service's databases so that restores can still be done automatically.

Modes

`bkreg` has two major modes: changing the contents of a backup table and displaying the contents of a backup table.

Changing Contents

`bkreg -p`	changes the rotation period for a backup table. The default rotation period is one week.
`bkreg -a`	adds an entry to a backup table.
`bkreg -e`	edits an existing entry in a backup table.
`bkreg -r`	removes an existing entry from a backup table.

Displaying Contents

`bkreg -C` produces a customized display of the contents of a backup table.
`bkreg [-A]`
`bkreg [-O]`
`bkreg [-R]` produces a summary display of the contents of a backup table.

Options

`-a` Adds a new entry to the default backup table. Options required with `-a` are: `-o`, `-c`, `-m`, and `-d`. If other options are not specified, the following defaults are used: the default backup table is used, no method options are specified, the priority is 0, and no dependencies exist between entries.

`-b` *moptions*
Each backup method supports a specific set of options that modify its behavior. *moptions* is specified as a list of options that are blank-separated and enclosed in quotes. The argument string provided here is passed to the method exactly as entered, without modification. [See incfile(AS_CMD), ffile(AS_CMD), fdisk(AS_CMD), fimage(AS_CMD), and fdp(AS_CMD) for lists of valid options.]

`-c` *weeks*`:`*days*
`-c demand`
Sets the week(s) and day(s) of the rotation period during which a backup entry should be performed or for which a display should be generated. *weeks* is a set of numbers between 1 and 52. The value of *weeks* cannot be greater than the value of *period*. *weeks* is specified as a combination of lists or ranges (either comma-separated or blank-separated and enclosed in quotes). An example set of weeks is

```
"1 3-10,13"
```

indicating the first week, each of the third through tenth weeks, and the thirteenth week of the rotation period. *days* is a set of numbers between 0 (Sunday) and 6 (Saturday) specified as a combination of lists or ranges (either comma-separated or blank-separated and enclosed in quotes). `demand` indicates that an entry is used only when explicitly requested by:

```
backup -c demand
```

[see backup(AS_CMD).]

`-d` *ddev*
Specifies *ddev* as the destination device for the backup operation. *ddev* is of the form:

dgroup`:`*ddevice*`:`*dchar*`:`*dlabels*

where either *dgroup* or *ddevice* must be specified and *dchar* and *dlabels* are optional. (Both *dgroup* and *ddevice* may be specified together.) Colons delineate field boundaries and must be included as indicated above. *dgroup* is the device group for the destination device. *ddevice* is the device name of a specific destination device. If *ddevice* is omitted, *dgroup* must be specified and any available device in *dgroup* may be used. *dchar* describes media characteristics. If specified, they override the default characteristics for the device and group. *dchar* is of the form:

keyword=value

where *keyword* is a valid device characteristic key word (as it appears in the device table). *dchar* entries may be separated by commas or blanks. If separated by blanks, the entire string of arguments to *ddev* must be enclosed in quotes. *dlabels* is a list of volume names of the destination volumes. The list of *dlabels* must be either comma-separated or blank-separated. If blank-separated, the entire *ddev* argument must be surrounded by quotes. Each *dlabel* corresponds to a *volumename* specified on the `labelit` command [see labelit in volcopy(AS_CMD)]. If *dlabels* is omitted, `backup` and `restore` [see restore(AS_CMD)] do not validate the volume labels on this entry.

`-e` Edits an existing entry. If any of the options `-b`, `-c`, `-d`, `-m`, `-o`, `-D`, or `-P` are present, they replace the current settings for the specified entry in the table.

`-h` Suppresses headers when generating displays.

`-m` [*method*]
Performs the backup using the specified `method`. Default methods are: `incfile`, `ffile`, `fdisk`, `fimage`, and `fdp`. If this is not a default method, it must appear as the executable file `/etc/bkup/method/METHOD`. `migration` indicates that the `-o` *orig* was a *ddev* during a prior backup operation. The originating object is not rearchived; it is simply copied to the `-d` *ddev* specified for the entry. The backup history (if any) and tables of contents (if any) are updated to reflect the changed destination for the original archive.

`-o` *orig*
Specifies *orig* as the originating object for the backup operation. *orig* is of the form:

oname`:`*odevice*[`:`*olabel*]

oname is an originating object name. For file system partitions, it is the nodename on which the file system is usually mounted [see mount(AS_CMD)]. For data partitions, it is any valid pathname. This value is provided to the backup method and validated by `backup`. The default data partition backup methods, `fdp` and `fdisk`, do not validate this name.

odevice is the device name for the originating object. In all cases, it is a raw disk partition device name.

olabel is the volume label for the originating object. For file system partitions, it corresponds to the *volumename* specified on the `labelit` command. A data partition may have an associated volume name. If it does, the name is only known externally (taped on the volume).

`-p` *period*
Sets the rotation period (in weeks) for the backup table to *period*. The minimum value is 1; the maximum value is 52. By default the current week of the rotation is set to 1.

`-r` Removes the specified entry from the table.

`-s` Suppresses wrap-around behavior when generating displays. Normal behavior is to wrap long values within each field.

`-t` *table*
Uses *table* instead of the default table, which is:
`/var/sadm/bkup/bkreg.tab`
Note: the table name "new" is disallowed.

`-v` Generates displays using (vertical) columns instead of (horizontal) rows. This allows more information to be displayed without encountering problems displaying long lines.

`-w` *cweek*
Sets the current week of the rotation period to *cweek*. *cweek* is an integer between 1 and the value of *period*. *cweek* cannot exceed *period*. The default is 1.

`-A` Displays a report describing all fields in the table. The display produced by this option is best suited as input to a filter, since in horizontal mode it produces extremely long lines.

`-C` *fields*
Generates a display of the contents of a backup table, limiting the display to the specified fields. The output is a set of lines, one per table entry. Each line consists of the desired fields, separated by a field separator character. *fields* is a list of field names (either comma-separated or blank-separated and enclosed in quotes) for the fields desired. The valid field names are `period`, `cweek`, `tag`, `oname`, `odevice`, `olabel`, `weeks`, `days`, `method`, `moptions`, `prio`, `depend`, `dgroup`, `ddevice`, `dchar`, and `dlabel`.

`-D` *depend*
Specifies a set of backup operations that must successfully complete before this operation may begin. *depend* is a list of *tag*(s) (either comma-separated or blank-separated and enclosed in quotes) naming the antecedent backup operations. A dependence may only exist between entries in the same table.

`-f` *c* Overrides the default output field separator. *c* is the character that will appear as the field separator on the display output. The default output field separator is colon (":").

`-O` Displays a summary of all originating objects with entries in the default backup table, or the specified backup table.

`-P` *prio*
Sets a priority of *prio* for this backup operation. The default priority is 0; the maximum priority is 100. Priority describes the relative priority of a backup operation with respect to the others. A backup operation is not started until all others with a higher priority are underway. All backup operations with the same priority may occur simultaneously, unless the priority is 0. All backups with priority 0 proceed sequentially in an unspecified order.

`-R` Displays a summary of all destination devices with entries in the specified table.

ERRORS

The exit codes for `bkreg` are the following:

0 = successful completion of the task

1 = one or more parameters to `bkreg` are invalid.

2 = an error has occurred which caused `bkreg` to fail to complete all portions of its task.

Errors are reported on standard error if any of the following occurs:

1. The *tag* specified in `bkreg -e` or `bkreg -r` does not exist in the backup table.
2. The *tag* specified in `bkreg -a` already exists in the table.

FILES

`/etc/bkup/bkreg.tab`	default backup table
`/var/sadm/bkup/`*table*`/bkreg.tab`	non-default backup tables
`/etc/bkup/method/*`	backup methods

EXAMPLE

Example 1:

```
bkreg -p 15 -w 3
```

establishes a 15-week rotation period in the default backup table and sets the current week to the 3rd week of the rotation period.

Example 2:

```
bkreg -a acct5 -t wklybu.tab \
-o /usr:/dev/rdsk/c1d0s2:usr \
-c "2 4-6 8 10:0,2,5" -m incfile -b -txE \
-d diskette:cap=1404:acctwkly1,acctwkly2,acctwkly3
```

adds an entry named `acct5` to the backup table named `wklybu.tab`. If `wklybu.tab` does not already exist, it will be created. The originating object to be backed up is the `/usr` file system on the `/dev/rdsk/c1d0s2` device which is known as `usr`. The backup will be performed each Sunday, Tuesday, and Friday of the second, fourth through sixth, eighth, and tenth weeks of the rotation period using the `incfile` (incremental file) method. The method options specify that a table of contents will be created on additional media instead of in the backup history log, the exception list is to be ignored, and an estimate of media usage for the archive is to be provided before performing the backup. The backup will be done to the next available diskette device using the three diskette volumes `acctwkly1`, `acctwkly2`, and `acctwkly3`. These volumes have a capacity of `1404` blocks each.

Example 3:

```
bkreg -e services2 -t wklybu.tab \
-o /back:/dev/rdsk/c1d0s8:back \
-m migration -c demand -d ctape:/dev/rdsk/c4d0s3
```

changes the specifications for the backup operation named `services2` on the backup table `wklybu.tab` so that whenever the command

```
backup –c demand
```

is executed, the backup that was performed to the destination device `back:dev/rdsk/c1d0s8:back` will be migrated from that device (now serving as the originating device) to a cartridge tape.

Example 4:

```
bkreg –e pubsfri –P 10 \
–D develfri,marketfri,acctfri
```

changes the priority level for the backup operation named `pubsfri` to 10 and makes this backup operation dependent on the three backup operations `develfri`, `marketfri`, and `acctfri`. Backup of `pubsfri` will occur only after all backup operations with priorities greater than 10 have begun and after backup operations for `develfri`, `marketfri`, and `acctfri` have completed successfully.

Example 5:

```
bkreg –c 1-8:0-6
```

provides the default display of the contents of the default backup table, for all weekdays for the first through eighth weeks of the rotation period. The display takes the following format:

```
Originating Device: /dev/dsk/c1d0s0
```

Tag	Weeks	Days	Method	Options	Priority	Dgroup
rootdai	1-8	1–6	incfile			diskette
rootsp	1-8	0	ffile	-bxt	20	ctape

```
Originating Device: /dev/rdsk/c1d0s2
```

Tag	Weeks	Days	Method	Options	Priority	Dgroup
usrdai	1-8	1–5	incfile			diskette
usrsp	1-8	0	ffile	-bxt	15	ctape

SEE ALSO

backup(AS_CMD), fdisk(AS_CMD), fdp(AS_CMD), ffile(AS_CMD), fimage(AS_CMD), incfile(AS_CMD), mkfs(AS_CMD), mount(AS_CMD), restore(AS_CMD), volcopy(AS_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.
Optional

NAME

bkstatus – display the status of backup operations

SYNOPSIS

`bkstatus` [`-a`] [`-h`] [`-f` *c*] [`-j` *jobids*] [`-u` *users*]

`bkstatus` [`-s` *states*] [`-h`] [`-f` *c*] [`-j` *jobids*] [`-u` *users*]

`bkstatus -p` *period*

DESCRIPTION

Without options, `bkstatus` displays the status of backup operations that are in progress: `active`, `pending`, `waiting` or `suspended`. When used with the `-a` option, `bkstatus` includes `failed` and `completed` backup operations in the display.

`bkstatus -p` defines the amount of status information, in weeks, that is saved for display.

`bkstatus` may only be executed by a user with appropriate privileges.

Each backup operation goes through a number of states as described below. The key letters listed in parentheses after each state are used with the `-s` option and also appear on the display.

`pending(p)` `backup` has been invoked and the operations in the backup table for the specified day are scheduled to occur.

`active(a)` The backup operation has been assigned a destination device and archiving is currently underway; or a suspended backup has been resumed.

`waiting(w)` The backup operation is waiting for operator interaction, such as inserting the correct volume.

`suspended(s)` The backup operation has been suspended by an invocation of `backup -S` [see backup(AS_CMD)].

`failed(f)` The backup operation failed or has been canceled.

`completed(c)` The backup operation has completed successfully.

The options and arguments have the following meanings:

`-a` Include `failed` and `completed` backup operations in the display. All backup operations that have occurred within *period* are displayed.

`-f` *c* Suppresses field wrap on the display and specifies an output field separator to be used. The value of *c* is the character that will appear as the field separator on the display output. For clarity of output, do not use a separator character that is likely to occur in a field. For example, do not use the colon as a field separator character if the display will contain dates that use a colon to separate hours from minutes. To use the default field separator (tab), specify the null character (`""`) for *c*.

`-h` Suppress header on the display.

`-j` *jobids* Restrict the display to the specified list of backup *jobids* [either comma-separated or blank-separated and enclosed in quotes; see backup(AS_CMD)].

`-p` *period* Define the amount of backup status information that is saved and made available for display as *period*. *Period* is the number of weeks that information is saved in `/etc/bkstatus.tab`. Status information that is older than the number of weeks specified in *period* is deleted from the status table. The minimum valid entry is 1. The maximum is 52. The default is 1 week.

`-s` *states* Restrict the report to backup operations with the specified *states*. *states* is a list of state key letters (concatenated, comma-separated or blank-separated and surrounded by quotes). For example,

```
apf
a,p,f
"a p f"
```

all specify that the report should only include backup operations that are `active`, `pending` or `failed`.

–u *users* Restrict the display to backup operations started by the specified list of *user*s (either comma-separated or blank-separated and enclosed in quotes).

ERRORS

The exit codes for `bkstatus` are the following:

0 = successful completion of the task

1 = one or more parameters to `bkstatus` are invalid.

2 = an error has occurred which caused `bkstatus` to fail to complete all portions of its task.

FILES

`bkstatus.tab` lists the current status of backups that have occurred or are still in progress.

`bkreg.tab` describes the backup policy decided on by the System Administrator.

`/etc/bkup/bkreg.tab` the default backup table.

EXAMPLE

Example 1:

```
bkstatus –p 4
```

specifies that backup status information is to be saved for four weeks. Any status information older than four weeks is deleted from the system.

Example 2:

```
bkstatus –a –j back-459,back-395
```

produces a display that shows status for the two backup jobs specified, even if they `completed` or `failed` for those jobs.

Example 3:

```
bkstatus –s a,c –u "oper3 oper4"
```

produces a display that shows only those backup jobs issued by users `oper3` and `oper4` that have a status of either `active` or `completed`.

SEE ALSO

backup(AS_CMD), bkhist(AS_CMD), bkreg(AS_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.

Optional

NAME

devnm – device name

SYNOPSIS

`devnm` *pathname*

DESCRIPTION

The command `devnm` identifies the special file associated with the mounted file system where the named file or directory resides. The full pathname must be given.

EXAMPLE

The command:

```
devnm /usr
```

produces

```
/dev/dsk/0s1 /usr
```

if `/usr` is mounted on `/dev/dsk/0s1`.

FILES

`/dev/dsk/*`
`/etc/mnttab`

USAGE

Administrator.

LEVEL

Level 1.

NAME

diskusg, acctdisk – generate disk accounting data by user ID

SYNOPSIS

`diskusg` [`-s`] [`-v`] [`-i` *fnmlist*] [`-p` *file*] [`-u` *file*] [*special-file* ...]

`acctdisk`

DESCRIPTION

The command `diskusg` generates disk accounting information for the file system identified by the *special-files*. `diskusg` prints lines on the standard output, one per user, in the following format:

uid login #blocks

where

uid is the numerical user ID of the user.

login is the login name of the user; and

#blocks is the total number of disk blocks allocated to this user.

The command `diskusg` is normally run in `dodisk` [see `dodisk` in acct(AS_CMD)].

The options and arguments have the following meanings:

`-s` The input data is already in `diskusg` output format. The command `diskusg` combines all lines for a single user into a single line.

`-v` Verbose; print a list on standard error of all files that are charged to no one.

`-i` *fnmlist* Ignore the data on those file systems whose file system name is in *fnmlist*. The argument *fnmlist* is a list of file system names separated by commas or enclose within quotes. The command `diskusg` compares each name in this list with the file system name stored in the volume ID [see `labelit` in volcopy(AS_CMD)].

`-p` *file* Generate login names from password file *file*. `/etc/passwd` is used by default.

`-u` *file* Write records to *file* of files that are charged to no one. Records consist of the special filename, the i-node number, and the user ID.

The command `acctdisk` expects a sequence of disk accounting information, as produced by `diskusg` (sorted by user ID and login name), and generates total accounting records that can be merged with other accounting records.

FILES

`/etc/passwd` used for user ID to login name conversions

`/usr/lib/acct` directory for accounting commands

USAGE

Administrator.

SEE ALSO

acct(AS_CMD), acct(KE_OS), volcopy(AS_CMD).

FUTURE DIRECTIONS

In the near future, file sizes will be reported in terms of a specific unit size, such as 1K, independent of the file system block size.

LEVEL

Level 2: September 30, 1989.

NAME

fsck – check and repair file systems

SYNOPSIS

`fsck` [`-F` *FSType*] [`-V`] [`-m`] [`-o` *specific_options*] [*special*`...`]

DESCRIPTION

The command `fsck` audits and interactively repairs inconsistent conditions in file systems. If the file system is found to be consistent, the number of files, blocks used, and blocks free are reported. Historically, if the file system is inconsistent the user is prompted for concurrence before each correction is attempted. It should be noted that some corrective actions may result in some loss of data. The amount and severity of data loss may be determined from the diagnostic output. Historically, the default action for each correction is to wait for the user to respond `yes` or `no`. If the user does not have write permission, `fsck` defaults to a `-n` action (see below).

FSType represents the file system type of the file system to be checked. *special* represents a special device (*e.g.*, `/dev/dsk/c1d0s8`). *specific_options* represent options specified as a comma-separated list of key words and/or key word attribute pairs which are to be interpreted by the *FSType*-specific module.

The options have the following meanings:

`-F` specify the *FSType* on which to operate. The *FSType* must be specified on the command line or must be determinable from an implementation-defined database. If the *FSType* is not specified on the command line, then the implementation-defined database must contain an entry for the special device.

`-o` specify *FSType*-specific options if any.

`-m` perform a sanity check only. `fsck` will return 0 if the file system is suitable for mounting. If the file system needs additional checking the return code is 32 and if the file system is mounted the return code is 33. Error codes larger than 33 indicate that the file system is badly damaged.

`-V` echo complete command line. The command line is generated by using the options and arguments provided plus determining the others by a lookup in an implementation-defined database. The command is not executed.

Historically, implementations have provided the following options:

`-p` correct inconsistencies that can be fixed automatically. These are inconsistencies that are deemed to be harmless and do not require confirmation by the administrator. Examples are unreferenced i-nodes, incorrect counts in the superblocks and missing blocks in the free list.

`-y` Assume a yes response to all questions asked by `fsck`.

`-n` Assume a no response to all questions asked by `fsck`; do not open the file system for writing.

`-s`*X* Ignore the actual free list and (unconditionally) reconstruct a new one. *X* is a hardware dependent option, which specifies how the free list is to be created; if it is not given, the values used when the file system was created, or other default values, are used.

`-S`*X* Conditionally reconstruct the free list. This option is like `-s`*X* above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using `-S` will force a no response to all questions asked by `fsck`. This option is useful for forcing free list reorganization on uncontaminated file systems.

`-t` *file* If `fsck` cannot obtain enough memory to keep its tables, it uses a scratch file. If the `-t` option is specified, *file* is used as the scratch file, if needed. Without the `-t` flag, `fsck` will prompt the user for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when `fsck` completes.

`-q` Quiet `fsck`. Unreferenced FIFOs will silently be removed. If `fsck` requires it, counts in the superblock will be automatically fixed and the free list salvaged.

`-D` Directories are checked for bad blocks. Useful after system crashes.

`-f` Fast check. Check block and sizes and check the free list. The free list will be reconstructed if it is necessary.

`-l` Identify badly damaged files by their logical name

Inconsistencies checked are as follows:

1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Incorrect number of blocks.
   Directory size not `{DIRSIZE}` aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated i-node.
   i-node number out of range.
8. Super Block checks:
   More than `{INODE_MAX}` i-nodes.
   More blocks for i-nodes than there are in the file system.
9. Bad free block list format.
10. Total free block and / or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the user's concurrence, reconnected by placing them in the `lost+found` directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. If it is empty, `fsck` will silently remove them. `fsck` will force the reconnection of nonempty directories. The name assigned is the i-node number.

Checking the raw device is almost always faster and should be used with everything but the `root` file system.

USAGE

Administrator.

The file system should be unmounted when `fsck` is used. If this is not possible, care should be taken that the system is quiescent and that the machine is rebooted immediately afterwards, if the file system is a critical one, for example, `root`.

SEE ALSO

mkfs(AS_CMD).

FUTURE DIRECTIONS

Support for the options `-p`, `-y`, `-n`, `-s`, `-S`, `-t`, `-q`, `-D`, `-f`, and `-l` will be discontinued in a future issue of the SVID.

LEVEL

Level 1.

The following options have been moved to level 2 effective September 30, 1989:

`-p`, `-y`, `-n`, `-s`, `-S`, `-t`, `-q`, `-D`, `-f`, and `-l`

NAME

fsdb – file system debugger

SYNOPSIS

`fsdb` [`–F` *FSType*] [`–V`] [`–o` *specific_options*] *special*

DESCRIPTION

The command `fsdb` is a file system debugger which allows for the manual repair of a file system after a crash. It is intended for experienced users only. *FSType* is the File System type of the special device to be debugged. *specific_options* are a comma separated list of key words and/or key word attribute pairs which are interpreted by the *FSType*-specific `fsdb`. *special* is the device (*e.g.* `/dev/dsk/c1d0s2`) on which the file system resides.

The options have the following meanings:

`–F` specify the *FSType* on which to operate. The *FSType* must be specified or must be determinable by searching an implementation-defined database for an entry matching the *special* specified.

`–o` specify *FSType*-specific options, if any.

`–V` echo complete command line. This includes additional information determined by a lookup in an implementation-defined database. This option is used to verify and validate a command line. The command is not executed.

USAGE

Administrator.

SEE ALSO

fsck(AS_CMD).

LEVEL

Level 1.

NAME

fstyp – determine file system type

SYNOPSIS

`fstyp` [`-v`] *special*

DESCRIPTION

The command `fstyp` allows the user to determine the file system type of unmounted file systems using heuristic programs.

`fstyp` invokes a number of file system type specific modules. Each of these modules applies some appropriate heuristic program to determine whether the supplied *special* file is of the type for which it checks. If it is, the program prints on standard output the usual file system type for that type and exits with a return code of 0; if none of the modules succeeds, the error message

```
unknown_fstyp (no matches)
```

is returned and the exit status is 1. If more than one module succeeds the error message

```
unknown_fstyp (multiple matches)
```

is returned and the exit status is 2.

The option has the following meaning:

`-v` Produce verbose output. Usually superblock information.

USAGE

Administrator.

WARNING: The use of heuristic programs implies that the result of *fstyp* is not guaranteed to be accurate.

In the case that multiple matches are found `fsck` should be run on the file system with the `-n` option with different *FSTypes* to determine the file system type [see fsck(AS_CMD)].

SEE ALSO

fsck(AS_CMD).

LEVEL

Level 1.

NAME

fuser – identify processes using a file or file structure

SYNOPSIS

/usr/sbin/fuser [-[c|f]ku] *files* **|** *resources* **[[-] [-[c|f]ku]** *files* **|** *resources***] . . .**

DESCRIPTION

fuser outputs the process IDs of the processes that are using the *files* or remote *resources* specified as arguments. Each process ID is followed by one of these letter codes, which identify how the process is using the file:

c as its current directory.

r as its root directory, which was set up by the **chroot**(SD_CMD) command.

o as an open file.

t as its text file.

For block special devices with mounted file systems, processes using any file on that device are listed. For remote resource names, processes using any file associated with that remote resource are reported. For all other types of files (text files, executables, directories, devices, and so on) only the processes using that file are reported.

The following options may be used with **fuser**:

-c may be used with files that are mount points for file systems. With that option the report is for use of the mount point and any files within that mounted file system.

-f when this is used, the report is only for the named file, not for files within a mounted file system.

-u the user login name, in parentheses, also follows the process ID.

-k the **SIGKILL** signal is sent to each process. Since this option spawns kills for each process, the kill messages may not show up immediately [see **kill**(BU_CMD)].

If more than one group of files are specified, the options may be respecified for each additional group of files. A lone dash cancels the options currently in force.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

Any user with permission to read **/dev/kmem** can use **fuser**. Only a privileged user can terminate another user's process.

EXAMPLES

fuser -ku /dev/dsk/1s?

if typed by a user with appropriate privileges, terminates all processes that are preventing disk drive one from being unmounted, listing the process ID and login name of each as it is killed.

fuser -u /etc/passwd
lists process IDs and login names of processes that have the password file open.

fuser -ku /dev/dsk/1s? -u /etc/passwd
executes both of the above examples in a single command line.

fuser -cu /home
if the **/dev/dsk/c1d0s9** device is mounted on **/home**, lists process IDs and login names of processes that are using **/dev/dsk/c1d0s9**.

FILES

/stand/unix	for system namelist
/dev/kmem	for system image
/dev/mem	also for system image

NOTE

If an RFS resource from a pre System V Release 4 server is mounted, **fuser** can only report on use of the whole file system, not on individual files within it. This RFS interface is no longer supported.

Because **fuser** works with a snapshot of the system image, it may miss processes that begin using a file while **fuser** is running. Also, processes reported as using a file may have stopped using it while **fuser** was running. These factors should discourage the use of the **-k** option.

fuser does not report all possible usages of a file (for example, a mapped file).

SEE ALSO

chroot (SD_CMD), **kill** (BU_CMD), **mount** (AS_CMD), **ps** (BU_CMD), **signal** (BA_OS)

LEVEL

Level 1.

NAME

fwtmp, wtmpfix – manipulate connect accounting records

SYNOPSIS

`fwtmp` [`–ic`]

`wtmpfix` [*files*]

DESCRIPTION

The command `fwtmp` reads from the standard input and writes to the standard output, converting binary records of the type found in `/var/adm/wtmp` to formatted readable records. The readable version is useful in editing bad records or general purpose maintenance of the file.

The option `–ic` is used to denote that input is in readable form, and output is to be written in binary form.

The command `wtmpfix` examines the standard input or named *files* in `wtmp` format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A – can be used in place of *files* to indicate the standard input. If time/date corrections are not performed, `acctcon1` [see `acctcon1` in acctcon(AS_CMD)] will fault when it encounters certain date-change records.

Each time the date is set, a pair of date change records are written to `/var/adm/wtmp`. The first record is the old date denoted by the string `old time` placed in the line field and the flag `OLD_TIME` placed in the type field of the `<utmp.h>` structure. The second record specifies the new date and is denoted by the string `new time` placed in the line field and the flag `NEW_TIME` placed in the type field. The command `wtmpfix` uses these records to synchronize all time stamps in the file.

In addition to correcting time/date stamps, `wtmpfix` will check the validity of the name field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to `INVALID` and write a diagnostic to the standard error. In this way, `wtmpfix` reduces the chance that `acctcon1` will fail when processing connect accounting records.

FILES

`/var/adm/wtmp`

USAGE

Administrator.

SEE ALSO

acct(AS_CMD), acct(KE_OS). acctcms(AS_CMD), acctcom(AS_CMD), acctcon(AS_CMD), acctmerg(AS_CMD), acctprc(AS_CMD), runacct(AS_CMD),

LEVEL

Level 2: September 30, 1989.

NAME

groupadd – add (create) a new group definition on the system

SYNOPSIS

`groupadd` [`-g` *gid* [`-o`]] *group*

DESCRIPTION

The command `groupadd` creates a new group definition on the system.

The options and arguments have the following meanings:

`-g` *gid* The group ID for the new group should be *gid*. It must be a non-negative integer less than `{MAXUID}`. It defaults to the next available, *i.e.*, unique, group ID above 99.

`-o` This option overrides the default unique UID enforcement and allows the *gid* to be non-unique.

group A unique string of printable characters that specifies the name of the new group. It may not include a colon (`:`).

FILES

`/etc/group`

SEE ALSO

groupdel(AS_CMD), groupmod(AS_CMD).

LEVEL

Level 1.

NAME

groupdel – delete a group definition from the system

SYNOPSIS

`groupdel` *group*

DESCRIPTION

The command `groupdel` deletes a group definition from the system.

The argument has the following meaning:

group A string of printable characters that specifies the group to be deleted. It must identify an existing group.

FILES

`/etc/group`

SEE ALSO

groupadd(AS_CMD), groupmod(AS_CMD).

LEVEL

Level 1.

NAME

groupmod – modify a group definition on the system

SYNOPSIS

`groupmod` [`-g` *gid* [`-o`]] [`-n` *name*] *group*

DESCRIPTION

The command `groupmod` modifies the definition of the specified group.

The options and arguments have the following meanings:

`-g` *gid* The new group ID for the *group* should be *gid*. It must be a non-negative integer less than `{MAXUID}`. It defaults to the next available, *i.e.*, unique, group ID above 99.

`-o` This option allows the *gid* to be non-unique, *i.e.*, overrides the default unique *gid* enforcement.

`-n` *name* A string of printable characters that specifies a new name for the *group*. It may not include a colon (`:`) and must be unique.

group The current name of the group to be modified. It must exist as a valid group.

SEE ALSO

groupadd(AS_CMD), groupdel(AS_CMD).

LEVEL

Level 1.

NAME

init – change system run level

SYNOPSIS

`init [0123456sSqQ]`

DESCRIPTION

The command `init` is used to direct the actions of the `init` process, which is the system process spawner. (The `init` command provides the `init` process with certain directives; it is important to keep in mind the distinction between the two.)

The system is in a particular run level at any given time. The current run level can be retrieved using `who -r` [see who(AU_CMD)]. The processes spawned by the `init` process for each of these run levels is defined in the `/etc/inittab` file. The system can be in one of eight run levels, `0–6` and `s` (or `S`). The run level is changed when the System Administrator runs the `init` command.

If the run level `s` (or `S`) is specified, the `init` process goes into the SINGLE-USER level. This is the only run level that does not require the existence of a properly formatted `/etc/inittab` file. (If that file does not exist, then by default the SINGLE-USER level is entered.)

If a run level of `0` through `6` is specified, the `init` process enters the corresponding run level.

The following arguments are accepted by `init`:

`0–6` tells `init` to place the system in one of the run levels `0–6`.

`q` (or `Q`) tells `init` to re-examine the `/etc/inittab` file. It is often used to check the correctness of that file after it has been changed.

`s` (or `S`) tells `init` to enter the SINGLE-USER level. When this level change is effected, the virtual system terminal, `/dev/console`, is changed to the terminal from which the command was executed.

FILES

`/etc/inittab`

USAGE

Administrator.

SEE ALSO

who(AU_CMD).

LEVEL

Level 1.

NAME

installf – add a file to the software installation database

SYNOPSIS

installf [-c *class***]** *pkginst pathname* **[***ftype* **[***major minor***]**
[*mode owner group***]]**

installf [-c *class***]** *pkginst* **-**

installf -f [-c *class***]** *pkginst*

installf [[-c *class] pkginst path1=path2* **[l|s]**

DESCRIPTION

installf is a tool available for use from within custom procedure scripts such as **preinstall**, **postinstall**, **preremove**, and **postremove**. When these scripts create or modify files, **installf** should be used to register the addition or change into the system's contents database.

When the second synopsis is used, the pathname descriptions will be read from standard input. These descriptions are the same as would be given in the first synopsis but the information is given in the form of a list. (The descriptions should be in the form: *pathname* [*ftype* [[*major minor*] [*mode owner group*]].)

When the last synopsis is invoked, the pathname argument is used to specify a link, where *path1* indicates the link and *path2* the file being linked to. The **ftypes l** and **s** are used to specify a hard link or symbolic link, respectively. If *ftype* is not specified, **installf** defaults to type 1.

After all files have been appropriately created and/or modified, **installf** should be invoked with the **-f** synopsis to indicate that installation is final. Links will be created at this time and, if attribute information for a pathname was not specified during the original invocation of **installf** or was not already stored on the system, the current attribute values for the pathname will be stored. Otherwise, **installf** verifies that attribute values match those given on the command line, making corrections as necessary. In all cases, the current content information is calculated and stored appropriately.

-c *class* Class with which installed objects should be associated. Default class is **none**.

pkginst Name of package instance with which the pathname should be associated.

pathname Pathname that is being created or modified. Special characters, such as an equal sign (**=**), are included in pathnames by surrounding the entire pathname in single quotes (as in, for example, **'/usr/lib/~='**). When a pathname is specified on a shell command line, the single quotes must be preceded by backslashes so they're not interpreted by the shell.

ftype A one-character field that indicates the file type. Possible file types include:

f a standard executable or data file
e a file to be edited upon installation or removal
v volatile file (one whose contents are expected to change)
d directory
x an exclusive directory
l linked file
p named pipe
c character special device
b block special device
s symbolic link

Once a file has the file type attribute **v**, it will always be volatile. For example, if a file being installed already exists and has the file type attribute **v**, then even if the version of the file being installed is not specified as volatile, the file type attribute will remain volatile.

major The major device number. The field is only specified for block or character special devices.

minor The minor device number. The field is only specified for block or character special devices.

mode The octal mode of the file (for example, 0664). A question mark (**?**) indicates that the mode will be left unchanged, implying that the file already exists on the target machine. If the directory doesn't exist, the default is 0755. If it's a file, the default is 0644. This field is not used for linked or symbolically linked files.

owner The owner of the file (for example, **bin** or **root**). The field is limited to 14 characters in length. A question mark (**?**) indicates that the owner will be left unchanged, implying that the file already exists on the target machine. If it doesn't exist, *owner* defaults to **root**. This field is not used for linked or symbolically linked files.

group The group to which the file belongs (for example, **bin** or **sys**). The field is limited to 14 characters in length. A question mark (**?**) indicates that the group will be left unchanged, implying that the file already exists on the target machine. If it doesn't exist, *group* defaults to **other**. This field is not used for linked or symbolically linked files.

-f Indicates that installation is complete. This option is used with the final invocation of **installf** (for all files of a given class).

EXAMPLES

The following example shows the use of **installf** invoked from an optional preinstall or postinstall script:

```
#create /dev/xt directory
#(needs to be done before drvinstall)
installf $PKGINST /dev/xt d 755 root sys ||
     exit 2
majno='/usr/sbin/drvinstall -m /etc/master.d/xt
     -d $BASEDIR/data/xt.o -v1.0' ||
     exit 2
i=00
```

```
while [$i -lt $limit]
do
  for j in 0 1 2 3 4 5 6 7
  do
    echo /dev/xt$i$j c $majno ‘expr $i * 8 + $j‘ 644 root sys
    echo /dev/xt$i$j=/dev/xt/$i$j
  done
  i=‘expr $i + 1‘
  [$i -le 9] && i="0$i" #add leading zero
done | installf $PKGINST - || exit 2
# finalized installation, create links
installf -f $PKGINST || exit 2
```

FILES

/usr/lib/locale/*locale*/**LC_MESSAGES/uxpkg**
language-specific message file [See **LANG** on **envvar**(BA_ENV).]

SEE ALSO

pkgadd (AS_CMD), **pkgask** (AS_CMD), **pkgchk** (AS_CMD), **pkginfo** (AS_CMD), **pkgmk** (AS_CMD), **pkgparam** (AS_CMD), **pkgproto** (AS_CMD), **pkgrm** (AS_CMD), **pkgtrans** (AS_CMD), **removef** (AS_CMD)

LEVEL

Level 1.

NOTICES

When *ftype* is specified, the required fields shown below must be defined:

ftype	Required Fields
p x d f v e	*mode owner group*
c b	*major minor mode owner group*

The **installf** command will create directories, named pipes and special devices on the original invocation. Links are created when **installf** is invoked with the **-f** option to indicate installation is complete.

For symbolically linked files, *path2* can be a relative pathname, such as **./** or **../**. For example, if you enter a line such as

```
installf -c none pkgx /foo/bar/etc/mount=../usr/sbin/mount s
```

path2 (**/foo/bar/etc/mount**) will be a symbolic link to **../usr/sbin/mount**.

When a link is specified, the directory in which the link is to reside must exist, otherwise **installf -f** will fail for that entry.

Files installed with **installf** will be placed in the class *none*, unless a class is defined with the command. Subsequently, they will be removed when the associated package is deleted. If this file should not be deleted at the same time as the package, be certain to assign it to a class which is ignored at removal time. To do this, associate the file to a class which will be handled by a removal class action script delivered with the package.

When classes are used, **installf** must be used as follows:

```
installf -c class1
installf -c class2
installf -f
```

Using multiple invocations is discouraged if standard input style invocations can be used with a list of files. This will be much faster because the **contents** file must be processed for each entry.

NAME

ipcrm – remove a message queue, semaphore set or shared memory ID

SYNOPSIS

`ipcrm` [`-q` *msqid*] [`-m` *shmid*] [`-s` *semid*] [`-Q` *msgkey*] [`-M` *shmkey*] [`-S` *semkey*]

DESCRIPTION

The command `ipcrm` will remove one or more specified message, semaphore or shared memory identifiers.

The options and arguments have the following meanings:

`-q` *msqid* Removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.

`-m` *shmid* Removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach operation.

`-s` *semid* Removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.

`-Q` *msgkey* Removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.

`-M` *shmkey* Removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach operation.

`-S` *semkey* Removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in `msgctl( )`, `shmctl( )`, and `semctl( )`. The identifiers and keys can be found by using `ipcs`.

SEE ALSO

ipcs(AS_CMD), msgctl(KE_OS), msgget(KE_OS), msgop(KE_OS), semctl(KE_OS), semget(KE_OS), semop(KE_OS), shmctl(KE_OS), shmget(KE_OS), shmop(KE_OS).

LEVEL

Level 1.

NAME

ipcs – report inter-process communication facilities status

SYNOPSIS

`ipcs` [*options*]

DESCRIPTION

The command `ipcs` prints certain information about active inter-process communication facilities. Note that information is displayed only for objects to which the user has read access. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the *options* and arguments for this command are as follows:

`-q` Print information about active message queues.

`-m` Print information about active shared memory segments.

`-s` Print information about active semaphores.

If any of the options `-q`, `-m`, or `-s` are specified, information about only those indicated is printed. If none of these three are specified, information about all three will be printed subject to these options:

`-b` Print maximum allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.

`-c` Print creator's login name and group name. See below.

`-o` Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

`-p` Print process number information. (Process ID of last process to send a message, process ID of last process to receive a message on message queues, process ID of creating process, process ID of last process to attach or detach on shared memory segments.) See below.

`-t` Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last `msgsnd()` and last `msgrcv()` operations [see `msgop`(KE_OS)] on message queues, last `shmat()` and last `shmdt()` operations [see `shmop`(KE_OS)] on shared memory, last `semop()` operation [see `semop`(KE_OS)] on semaphores.) See below.

`-z` Print alias name of security level. This option is only valid if the Enhanced Security Extension is implemented.

`-Z` Print fully qualified security level. This option is only valid if the Enhanced Security Extension is implemented.

`-a` Use all print *options*. (This is a shorthand notation for `-b`, `-c`, `-o`, `-p`, `-t`, and `-Z`.)

`-C` *corefile* The argument is taken as the name of an alternate *corefile* file, to be used instead of the default.

`-N` *namelist*
The argument is taken as the name of an alternate *namelist* file, to be used instead of the default.

The `-z` and `-Z` options are mutually exclusive. If the `-z` option is specified and there is not an alias assigned to the level, the decimal value of the level identifier (LID) is displayed. If the `-z` or `-Z` option is specified and the level is in the *valid-inactive* state, the decimal value of the LID is displayed. LID states are described in `lvlname`(ES_CMD).

The column headings and the meaning of the columns in an `ipcs` listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; `all` means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do not determine which facilities will be listed.

`T` (all) Type of facility:

- `q` message queue
- `m` shared memory segment
- `s` semaphore

`ID` (all) The identifier for the facility entry.

`KEY` (all) The key used as an argument in calls to `msgget()`, `semget()`, or `shmget()` to create the facility entry. (Note: The key of a shared memory segment is changed to `IPC_PRIVATE` when the segment has been removed until all processes attached to the segment detach it.)

`MODE` (all) The facility access modes and flags. The mode consists of 11 characters, interpreted as follows.

The first character is:

- `S` if a process is waiting on a `msgsnd()` operation;
- `D` if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it.
- `-` if neither of the above are true.

The second character is:

- `R` if a process is waiting on a `msgrcv()` operation;
- `C` if the associated shared memory segment is to be cleared when the first attach operation is executed.
- `-` if neither of the above are true.

The next nine characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next, to permissions of others in the user-group of the facility entry; and the last, to all others. Within each set, the first character

indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

- `r` if read permission is granted.
- `w` if write permission is granted.
- `a` if alter permission is granted.
- `–` if the indicated permission is not granted.

`OWNER`	(all)	The login name of the owner of the facility entry.
`GROUP`	(all)	The group name of the owner of the facility entry.
`LEVEL`	(all)	The level identifier of the facility entry.
`CREATOR`	(a,c)	The login name of the creator of the facility entry.
`CGROUP`	(a,c)	The group name of the creator of the facility entry.
`CBYTES`	(a,o)	The number of bytes in messages currently outstanding on the associated message queue.
`QNUM`	(a,o)	The number of messages currently outstanding on the associated message queue.
`QBYTES`	(a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue.
`LSPID`	(a,p)	The process ID of the last process to send a message to the associated queue.
`LRPID`	(a,p)	The process ID of the last process to receive a message from the associated queue.
`STIME`	(a,t)	The time the last message was sent to the associated queue.
`RTIME`	(a,t)	The time the last message was received from the associated queue.
`CTIME`	(a,t)	The time the associated entry was created or changed.
`NATTCH`	(a,o)	The number of processes attached to the associated shared memory segment.
`SEGSZ`	(a,b)	The size of the associated shared memory segment.
`CPID`	(a,p)	The process ID of the creator of the shared memory entry.
`LPID`	(a,p)	The process ID of the last process to attach or detach the shared memory segment.
`ATIME`	(a,t)	The time the last attach on the associated shared memory segment was completed.
`DTIME`	(a,t)	The time the last detach on the associated shared memory segment was completed.

`NSEMS`	(a,b)	The number of semaphores in the set associated with the semaphore entry.
`OTIME`	(a,t)	The time the last semaphore operation on the set associated with the semaphore entry was completed.

FILES

`/stand/unix`	system namelist
`/dev/kmem`	memory
`/etc/passwd`	user names
`/etc/group`	group names

USAGE

Things can change while `ipcs` is running; therefore the status it reports may no longer be accurate at the time it is seen.

SEE ALSO

lvlname(ES_CMD), msgop(KE_OS), semop(KE_OS), shmop(KE_OS).

LEVEL

Level 1.

NAME

killall – kill all active processes

SYNOPSIS

`killall` [*signal*]

DESCRIPTION

The command `killall` is a procedure used to kill all active processes not directly related to the `killall` procedure.

The command `killall` is chiefly used to terminate all processes with open files, so that the mounted file systems will be unbusied and can be unmounted.

The command `killall` sends *signal* to all remaining processes not belonging to the above group of exclusions. If no *signal* is specified, `SIGKILL` is used.

USAGE

Administrator.

SEE ALSO

kill(BA_OS), kill(BU_CMD), signal(BA_ENV).

LEVEL

Level 1.

NAME

last – indicate last logins by user or terminal

SYNOPSIS

`last` [`-` *number*] [`-f` *filename*] [*name/tty*] ...

DESCRIPTION

The command `last` looks back in the `wtmp` file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example `last 0` is the same as `last tty0`. If multiple arguments are given, the information which applies to any of the arguments is printed. For example `last root console` would list all of ''root's'' sessions as well as all sessions on the console terminal. `last` displays the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, `last` so indicates.

The pseudo-user `reboot` logs in at reboots of the system, thus

```
last reboot
```

will give an indication of mean time between reboot.

`last` with no arguments displays a record of all logins and logouts, in reverse order.

If `last` is interrupted, it indicates how far the search has progressed in `wtmp`. If interrupted with a quit signal (generated by a `CTRL-\`) `last` indicates how far the search has progressed so far, and the search continues.

`-` *number* limit the number of entries displayed to that specified by *number*.

`-f` *filename* Use *filename* as the name of the accounting file instead of `/var/adm/wtmp`.

FILES

`/var/adm/wtmp` login data base

LEVEL

Level 1.

NAME

link, unlink – exercise link and unlink system calls

SYNOPSIS

`link` *file1 file2*

`unlink` *file*

DESCRIPTION

The commands `link` and `unlink` perform their respective system calls on their arguments, without any error checking.

FILES

`/usr/sbin` commands directory.

USAGE

Only a user with appropriate privileges may execute these commands.

SEE ALSO

link(BA_OS), unlink(BA_OS).

FUTURE DIRECTIONS

`link` and `unlink` will be removed from the SVID when the Level 2 period has elapsed. Their functionality has been replaced by the `ln` command.

LEVEL

Level 2, July 1992.

NAME

logins – list user and system login information

SYNOPSIS

`logins` [`-abdhmopstuvx`][`-g` *groups*][`-l` *logins*]

DESCRIPTION

The command `logins` displays information on user and system logins. Content of the output is controlled by the command options and can include: user or system login, user ID number, `/etc/passwd` account field value (user name or other information), primary group name, primary group ID, multiple group names, multiple group IDs, home directory, login shell, user security level, user audit events, and four password aging parameters. The default information is: login ID, user ID, primary group name, primary group ID and the account field value from `/etc/passwd`. Output is sorted by user ID, displaying system logins followed by user logins.

The options and arguments have the following meanings:

`-a` Adds two password expiration fields to the display. The fields show how many days a password can remain unused before it automatically becomes inactive and the date that the password will expire.

`-b` Prints the user's auditable events. This option is only valid when the Auditing Extension is implemented.

`-d` Selects logins with duplicate UIDs.

`-h` Prints the valid login levels for the users. The levels are displayed one per line, with the default level first. This option is only valid when the Enhanced Security Extension is implemented.

`-m` Displays multiple group membership information.

`-o` Formats output into one line of colon-separated fields.

`-p` Selects logins with no passwords.

`-s` Selects all system logins.

`-t` Sorts output by login instead of by UID.

`-u` Selects all user logins.

`-v` Prints the user's default login level. This option is only valid when the Enhanced Security Extension is implemented.

`-x` Prints an extended set of information about each selected user. The extended information includes home directory, login shell and password aging information, each displayed on a separate line. The password information consists of password status (`PS` for passworded, `NP` for no password or `LK` for locked). If the login is passworded, status is followed by the date the password was last changed, the number of days required between changes, and the number of days allowed before a change is required.

`-g` *groups* Selects all users belonging to *groups*, sorted by login. Multiple groups can be specified as a comma-separated list.

`-l` *logins* Selects the requested *logins*. Multiple logins can be specified as a comma-separated list.

FILES

```
/etc/security/ia/master
/etc/passwd
/etc/group
```

USAGE

Options may be used together. If so, any login matching any criteria will be displayed. When the `-l` and `-g` options are combined, a user will only be listed once, even if they belong to more than one of the selected groups.

SEE ALSO

passwd(AU_CMD), useradd(AS_CMD), usermod(AS_CMD), userdel(AS_CMD).

LEVEL

Level 1.

NAME

mkfifo – make FIFO special file

SYNOPSIS

`mkfifo` *path ...*

DESCRIPTION

The command `mkfifo` creates the FIFO special files named by its argument list. The arguments are taken sequentially, in the order specified; and each FIFO special file is either created completely or, in the case of an error or signal, not created at all.

For each *path* argument, the `mkfifo` command behaves as if the function `mkfifo()` [see mkfifo(BA_OS)] was called with the argument *path* set to *path* and the *mode* set to the bitwise inclusive OR of `S_IRUSR`, `S_IWUSR`, `S_IRGRP`, `S_IWGRP`, `S_IROTH` and `S_IWOTH`.

If errors are encountered in creating one of the special files, `mkfifo` writes a diagnostic message to the standard error and continues with the remaining arguments, if any.

ERRORS

The command `mkfifo` returns exit code `0` if all FIFO special files were created normally; otherwise it prints a diagnostic and returns a value greater than `0`.

SEE ALSO

mkfifo(BA_OS).

LEVEL

Level 1.

NAME

mkfs – construct a file system

SYNOPSIS

`mkfs` [`-F` *FSType*] [`-V`] [`-m`] [*current_options*] [`-o` *specific_options*] *special* [*operands*]

DESCRIPTION

The command `mkfs` constructs a file system by writing on *special*; *special* must be the first argument after the options are given. The file system is created based on the *FSType*, *specific_options* and *operands* specified on the command line. If the *FSType* is not specified using the `-F` option, it is determined by a `/etc/vfstab` lookup by matching entries based on the *special* device.

The `mkfs` command waits 10 seconds before starting to construct the file system. During this time the command can be aborted by entering a delete (DEL).

The options and arguments have the following meanings:

`-F` *FSType* — Specify the *FSType* to be constructed. The *FSType* must be specified or must be determinable by a `/etc/vfstab` lookup by matching an entry on the *special* specified.

`-V` — Echo complete command line. This includes additional information as determined by an `/etc/vfstab` lookup. It can be used to verify and validate the command line. The command is not actually executed.

`-m` — Return the command line which was used to create the file system. The file system must already exist and this option provides a means of determining the command used in constructing the file system. It cannot be used with *current_options*, *specific_options*, or *operands*. It must be invoked by itself.

`-o` *specific_options* — Specify *FSType*-specific options, if any. *specific_options* are a list of keywords and/or keyword-attribute pairs (separated by commas) which are interpreted by the *FSType*-specific module of `mkfs`.

The *current_options* are options supported by the s5-specific module of `mkfs`. Other *FSTypes* do not necessarily support these options.

The *operands* are *FSType*-specific.

ERRORS

File systems do not need to be constructed for all *FSTypes*. Specifying these *FSTypes* on the command line will cause `mkfs` to fail.

The `mkfs` command does not determine whether the special file is block special. Some file system types require that it be a block special device whereas network file system types do not have this constraint.

FILES

`/etc/vfstab` default table of file system information

USAGE

Administrator.

SEE ALSO

makefsys(AS_CMD).

LEVEL

Level 1.

NAME

mkmsgs – create message files for use by gettxt

SYNOPSIS

`mkmsgs` [`-o`] [–i *locale*] *inputstrings msgfile*

DESCRIPTION

The `mkmsgs` utility is used to create a file of text strings that can be accessed using the text retrieval commands and function [see gettxt(BU_CMD), srchtxt(AS_CMD), and gettxt(BA_LIB)]. It will take as input a file of text strings for a particular geographic locale [see setlocale(BA_OS)] and create a file of text strings in a format that can be retrieved by the `gettxt` command and `gettxt()` routine. By using the `-i` option, the created file can be installed in the `/usr/lib/locale/`*locale*`/LC_MESSAGES` directory (*locale* corresponds to the language in which the text strings are written).

The options and arguments have the following meanings:

inputstrings the name of the file that contains the original text strings.

msgfile the name of the output file where `mkmsgs` writes the strings in a format that is readable by the command `gettxt` and the routine `gettxt()`. *msgfile* can be up to 14 characters in length, but may not contain \ 0 (null) or the ASCII code for `/` (slash) or `:` (colon).

`-i` *locale* install *msgfile* in the `/usr/lib/locale/`*locale*`/LC_MESSAGES` directory. Only a user with appropriate privileges, such as a member of the group `bin`, can create or overwrite files in this directory. The command should be run at **`SYS_PUBLIC`** level so that message files will be accessible to applications. Directories under `/usr/lib/locale` will be created if they don't exist.

`-o` overwrite *msgfile*, if it exists.

The input file contains a set of text strings for the particular geographic locale. Text strings are separated by a newline character. Nongraphic characters must be included as alphabetic escape sequences. Messages are transformed and copied sequentially from *inputstrings* to *msgfile*. To generate an empty message in *msgfile*, leave an empty line at the correct place in *inputstrings*.

Strings can be changed simply by editing the file *inputstrings*. New strings must be added only at the end of the file; then a new *msgfile* file must be created and installed in the correct place. If this procedure is not followed, the retrieval function will retrieve the wrong string and software compatibility will be broken.

The messages in the *inputstrings* must be in the following form:

```
string1
string2
 .
 .
 .
```

EXAMPLE

The following example shows an input message source file

```
File %s:\t cannot be opened\n
%s: Bad directory\n
      .
      .
      .
write error\n
      .
      .
```

The following command uses the input strings from `C.str` to create text strings in the appropriate format on the file `UX` in the current directory:

```
mkmsgs C.str UX
```

The following command uses the input strings from `FR.str` to create text strings in the appropriate format on the file `UX` in the directory `/usr/lib/locale/french/LC_MESSAGES/UX`.

```
mkmsgs –i french FR.str UX
```

FILES

`/usr/lib/locale/`*locale*`/LC_MESSAGES/*` message files created by `mkmsgs`

SEE ALSO

gettxt(BA_LIB), gettxt(BU_CMD), setlocale(BA_OS), srchtxt(AS_CMD).

LEVEL

Level 1.

NAME

mknod – build special file

SYNOPSIS

mknod *name* b *major minor*

mknod *name* c *major minor*

mknod *name* p

DESCRIPTION

The command mknod makes a directory entry and corresponding i-node for a special file.

The command mknod can also be used to create FIFOs (named pipes) (third case in **SYNOPSIS** above).

The first argument, *name*, is the name of the entry. The second argument is b if the special file is block-type (disks, tape) or c if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (*e.g.*, unit, drive, or line number), which may be either decimal or octal (any number with a leading zero).

The assignment of major device numbers is specific to each system.

The command mknod may only be used by a user with appropriate privileges, to make special files.

USAGE

Administrator.

SEE ALSO

mknod(BA_OS).

LEVEL

Level 1.

NAME

modadmin – loadable kernel module administration

SYNOPSIS

modadmin -l *modname . . .* **|** *pathname . . .*

modadmin -u *modid . . .*

modadmin -U *modname . . .*

modadmin -q *modid . . .*

modadmin -Q *modname . . .*

modadmin -s | S

modadmin -d *dirname* **| D**

DESCRIPTION

modadmin is the administrative command for loadable kernel modules. It performs the following functions:

load a loadable module into a running system

unload a loadable module from a running system

display the status of a loadable module(s) that is currently loaded

modify the loadable modules search path

The loadable modules feature lets you add a module to a running system without rebooting the system or rebuilding the kernel. When the module is no longer needed, this feature also lets you dynamically remove the module, thereby freeing system resources for other use.

Types of modules that can be dynamically loaded include:

device drivers (block, character, STREAMS and pseudo)

Host Bus Adapter (HBA) drivers

Direct Coupled Device (DCD) controller drivers

STREAMS modules

file systems

exec modules

system calls

miscellaneous modules, such as modules containing code for support routines shared among multiple loadable modules which are not needed in the statically configured kernel

Loadable modules are maintained in individual object files (**.o** files) in the same manner as statically configured modules. Unlike static modules, loadable modules:

are not linked to the kernel until they are needed

must be configured into the system and registered with the running kernel

must be configured in loadable form (requires writing additional module initialization or ''wrapper'' code)

can be loaded and unloaded by the user, as required, using the **modadmin** command, or by **init**(AS_CMD), during every system reboot,

can be loaded and unloaded by the kernel itself (called an auto load / unload—see NOTICES section)

The **modadmin** options have the following meanings:

-l *modname* Load the named module(s) *modname* using the current value of the loadable modules search path to locate the module's object file on disk.

This option searches for a matching file in all directories specified in the loadable modules search path. By default, the search pathname is **/etc/conf/mod.d**.

The load operation performs all tasks associated with link editing the module to the kernel and making the module accessible to the system. If the module depends on other loadable modules (as defined in **/etc/conf/mdevice.d**), and these modules are not currently loaded, **modadmin** will automatically load the dependent modules during the load operation.

When loading completes, an integer *modid* prints on the standard output to identify the module(s) that was loaded.

-l *pathname* Same as **-l** *modname*, except the absolute pathname *pathname* is used to locate the module's object file.

-u *modid* Unload the module(s) identified by the integer value *modid*.

If *modid* specifies **0** (zero), **modadmin** attempts to unload all loadable modules.

The unload operation performs all tasks associated with disconnecting the module from the kernel and releasing any memory acquired by the module. When unloading completes, an integer *modid* prints on the standard output to identify the module(s) that was unloaded.

If the module(s) to be unloaded are currently in-use, are dependents of a loadable module that is currently loaded, or are currently being loaded or unloaded, the unload request will fail.

-U *modname* Same as **-u** *modid*, except the module(s) to be unloaded is specified by name *modname*.

-q *modid* Print the status of the loaded module(s) identified by the integer value *modid*.

Information returned by this option includes:

module identifier (*modid*)

the module's pathname

the module's virtual load address

amount of memory the module occupies

the module's reference count

the module's dependent count

the module's unload delay value

the module's descriptive name

the type of module

depending on the type of module, either the module's character major number, block major number, file system switch number, or STREAMS switch number

-Q *modname* Same as **-q** *modid*, except the module(s) for which status information is to be reported is specified by name *modname*.

-s Print an abbreviated status for all modules currently loaded.

This option returns a listing of module names and module identifiers only.

-S Print the full status for all modules currently loaded.

This option returns status information of the form returned by the **-q** option.

-d *pathname* Prepend the pathname *pathname* to the current loadable modules search path, where *pathname* specifies directories that should be searched:

for all subsequent demand loads initiated by a **modadmin** command with the option **-l** and a named *modname*

for all subsequent loads performed by the kernel's autoload mechanism (see NOTICES section)

prior to searching any directories already prepended to the search path by a prior **modadmin** command with the option **-d**

prior to searching the default search path **/etc/conf/mod.d**, which is always searched, and is always searched last

pathname must specify an absolute pathname or a list of absolute pathnames delimited by colons. The directories identified by *pathname* do not have to exist on the system at the time the request to modify the search path using **modadmin** is made. If these directories do not exist at the time a load takes place, the load operation ignores them.

All modifications to the search path made using this option take effect immediately and affect all subsequent loads (demand and auto-load) and all users on the system.

-D Reset the loadable modules search path to its default value, **/etc/conf/mod.d**. The reset takes effect immediately and affects all subsequent loads (demand and auto-load) and all users on the system.

Errors

In the following conditions, **modadmin** fails and sets **errno** to:

ENOLOAD
failure in loading a loadable exec module

NOTICES

Auto Loading

Auto-load occurs when the kernel detects a particular loadable module is required to accomplish some task, but is not currently loaded. For example, if the task were a mount of a file system, and the loadable module that supports the file system was not loaded, the kernel would automatically load the file system module. Once the module was loaded, the mount would take place.

Auto-unload occurs when the kernel detects that the amount of available memory is low. At this time, the kernel begins unloading all modules that are not currently in-use—and that have not been used for some predetermined amount of time—to reclaim the memory allocated to these modules. Unloading continues until the amount of available memory reaches a predetermined high water mark, or the list of modules that are candidates for unloading is exhausted. The amount of time that must elapse before unused modules are considered candidates for unloading is controlled by the value of the global tunable parameter **DEF_UNLOAD_DELAY** in **/etc/conf/mtune.d**. Individual modules can override the value of the global auto-unload delay by specifying their own auto-unload delay value in their **Mtune** files.

Modules that are demand loaded using the **modadmin** command cannot be auto unloaded by the kernel. If a demand-loaded module is no longer needed in the system, it must be demand-unloaded. If a demand unload for a loaded module fails (because the module is in-use, for example) the unload mechanism will add the module to a list of modules that are candidates for the next auto-unload.

Loadable HBA Driver Considerations

Loadable HBA drivers:

must be demand loaded by the user via the **modadmin** command, or (during system reboot) demand loaded by **init** via the **idmodload** command

can not be auto loaded

can not be unloaded (demand or auto)

Loadable DCD Controller Driver Considerations

Loadable DCD controller drivers can not be demand loaded. They are auto loaded / unloaded by the kernel as required.

System Profiler

When the system profiler **prf** is turned on, loadable modules are locked into memory and cannot be unloaded. Modules can continue to be loaded with profiling enabled, but these modules will also become locked. When profiling is disabled, the locks for all loadable modules are removed.

SEE ALSO

init(AS_CMD), **modload**(KE_OS), **moduload**(KE_OS), **modpath**(KE_OS), **modstat**(KE_OS)

LEVEL

Level 1.

NAME

mount, umount – mount or unmount file systems and remote resources

SYNOPSIS

```
mount [-v]
mount [-p]
mount [-z]
mount [-Z]
mount [-llevel] [-F FSType] [-V] [-r] [-o specific_options] special
mount [-llevel] [-F FSType] [-V] [-r] [-o specific_options] mount_point
mount [-llevel] [-F FSType] [-V] [-r] [-o specific_options] special mount_point

umount [-V] [-o specific_options] special
umount [-V] [-o specific_options] mount_point
```

DESCRIPTION

File systems other than `root (/)` are considered removable in the sense that they can be either available to users or unavailable. The command `mount` makes available to users *special*, a block special device, or a remote resource from the *mount_point*. The *mount_point* must already exist; it then becomes the name of the root of the newly mounted *special* or remote resource. A unique resource may be mounted only once (no multiple mounts).

When entered with arguments, `mount` validates all arguments except for the *special* device and invokes an *FSType*-specific `mount` module. If invoked with no arguments, `mount` lists all the mounted file systems from the mount table. If invoked with *special*, *mount_point* or both arguments but without *FSType*, then `mount` will search `/etc/vfstab` to fill in the missing arguments: *FSType*, *special*, *specific_options*, *mount_point*, or *level*. The *level* field is populated only when the Enhanced Security Extension is implemented. It will then invoke the *FSType*-specific `mount` module.

When the Enhanced Security Extension is implemented, if *special* is a block special device, then the MAC level of the *mount_point* must be enclosed by the device (*special*) level range stored in the Device Database. The root level of the new mounted file system is set to the level of the *mount_point* and the file system level floor and ceiling are initialized to the level of the *mount_point*. If the `-l` option is invoked, then the level ceiling of the mounted file system is set to the *level* specified, after verifying that this level matches or dominates the level floor of the file system. Otherwise, if the `vfstab` is queried by the `mount` command and a level ceiling is stored in the `vfstab` for that entry, then the level ceiling of the mounted file system is set to the level stored in the `vfstab` after verifying that this level matches or dominates the level floor of the mounted file system. The mounted file system level range restricts the creation of file system objects to be within that level range.

Most *FSTypes* do not have a `umount` specific module. If one exists it is executed; otherwise the generic will unmount the file systems. If the `-o` option is specified, the `umount` specific module is always executed.

special indicates the block special device that is to be mounted on *mount_point*. It may also be a remote resource. *mount_point* indicates the mount point where *special* will be mounted. The *mount_point* is a directory which must already exist.

The options and arguments have the following meanings:

`-v` Prints output in a way similar to `mount` with no options or arguments, except that the *FSType* and flags are displayed, and the *mount_point* and *special* fields are reversed.

`-p` Prints the list of mounted file systems in the `vfstab` format.

`-z` Displays the alias level of the level ceiling of each mounted file system. This option is only valid if the Enhanced Security Extension is implemented.

`-Z` Displays the fully qualified level of the level ceiling of each mounted file system. This option is only valid if the Enhanced Security Extension is implemented.

`-l` *level* Sets the level ceiling of the mounted file system. The *level* must be either a valid security level alias or a valid fully qualified level name in the format:

h_name[:*c_name*[,*c_name*]...]

where *h_name* is a hierarchical classification name and *c_name* is a non-hierarchical category name. A fully qualified level is valid if the classification and categories compromising the level are named and if the level has been assigned a system level identifier (LID) using the `lvlname` command. An alias name is valid if the alias has been assigned to a fully qualified level using the `lvlname` command.

The *level* must be dominated (meaning equal to or greater than) by the high range of the device on which the file system is to be mounted. The *level* must also match or dominate the low range of the device. The level range of the device is stored in the Device Database (DDB). Note that the low range of the file system is the level of the mount point on which the file system is to be mounted.

This option is only valid when the Enhanced Security Extension is implemented.

`-F` *FSType* Specify the *FSType* on which to operate. The *FSType* must be specified or must be determinable from `/etc/vfstab` while mounting the file system.

`-V` Echo the complete command line. This includes additional information as determined by an `/etc/vfstab` lookup. This option can be used to verify and validate a command line. The command is not actually executed.

`-r` Indicates that *special* is to be mounted read-only. If *special* is write-protected or read-only, this flag must be used.

`-o` *specific_options* Specify *FSType*-specific options, if any. *specific_options* are a list of keywords and/or keyword-attribute pairs (separated by commas) which are interpreted by the *FSType*-specific module of `mount/umount`.

`mount` can be used by any user to list mounted file systems and resources. Only a process with appropriate privileges can mount or unmount file systems.

The `-z` and `-Z` options are mutually exclusive. If the `-z` option is specified and there is not an alias assigned to the level, the decimal value of the level identifier (LID) is displayed. If the `-z` or `-Z` option is specified and the level is in the *valid-inactive* state, the decimal value of the LID is displayed. LID states are described in `lvlname`(ES_CMD).

FILES

`/etc/mnttab`	mount table
`/etc/vfstab`	table of file system information

USAGE

Administrator.

SEE ALSO

lvlname(ES_CMD), mount(BA_OS), umount(BA_OS), setmnt(AS_CMD).

FUTURE DIRECTIONS

The old output format will be phased out in a future release and all output will be in the new –`v` format. The most significant changes are in the addition of two new fields to show the *FSType* and flags and the reversal of the *mount_point* and *special* name.

Support for the `-r` option will be removed in a future issue of the SVID.

LEVEL

Level 1.

The following option has been moved to level 2 effective September 30, 1989:

`-r`

NAME

msgalert – message alerting facility

SYNOPSIS

msgalert –a [–c *classification***[,***classification* **...]] [–l** *label***]**
[–s *severity***[,***severity* **...]]**
[–m *msg_text***] [–n** *num***] [–p** *period***] [–S** *time***] [–E** *time***] [–e** *command***]**
[–d *system***[,***system* **...]]** *ident*

msgalert –r | **–R** *ident* **...**

msgalert [–o] [*ident* **...]**

msgalert [–b on |**off] [–g on** |**off]**

DESCRIPTION

msgalert is a tool for monitoring kernel messages and standard format messages and directing alerts to the console of the local or a remote system. A standard format message is a message defined by a programmer in an application program and sent to the alerting facility via the **lfmt** command, or the **lfmt** or **vlfmt** library routines. An alert is itself a standard format message, and includes a date/time stamp, the system name, and the text of the message that generated the alert. For example:

UX:logalert_proc:WARNING: alert condition at *date/time* **on** *system* **-**
whole message generating the alert gets displayed here (bells)

Monitoring of messages is enabled or disabled using the **–g** option. An alerting request is created using the **–a** option; it can be restricted to a subset of messages by specifying any combination of the options which identify components of standard format messages. When an alerting request is created, monitoring begins immediately by default, and when the request's criteria are matched *num* times, an alert is generated. Following that, an alert is generated for each successive occurrence of the message until one of the following occurs:

- the criteria are no longer met (that is, *period* or the time frame elapses without the message reoccurring).
- the administrator removes the request causing the alert to be generated.
- the administrator resets the request causing the alert to be generated.

An alerting request is removed from the current set of requests using the **–r** option. An alerting request is reset using the **–R** option. That is, the request remains in the current set of requests, but its occurrence and period tallies are reset to zero.

If the Remote Operation Interface is available on both the local and remote system(s), the administrator on the local system can use the **–d** option to generate alerts on the remote system(s) for messages occurring on the local system.

When the **–o** option, or no option, is specified, a report of the current state of the alerting facility and a list of the current set of alerting requests is displayed. Specifying *ident* produces a report on only that alerting request.

msgalert has the following options and arguments.

-o Omit headers from the report of the current set of alerting requests and display the output in data format (fields are delimited by a semicolon, multiple values in a field are delimited by a comma).

-r Remove alerting request *ident* from the current set.

-R Reset *period* to be the start of a new period, and *num* to be zero (**0**) for *ident*. *ident* must be an existing message alerting request.

-b on| **off** Enable / disable the two-bell signal that is generated as part of the message alert. By default this option is set to **on**.

-g on| **off** Enable / disable the message alerting facility. By default this option is set to **off**.

-a Add alerting request *ident* to the current set of alerting requests. By default, the set is empty. See **lfmt**(BA_OS) for complete information on the standard format message components *classification*, *label*, *severity*, and *msg_text*. The following options can be used with **-a**:

- **-c** *classification* Specify the classification of messages to be monitored. Acceptable values can be one from each of the keyword sets [**soft** | **hard** | **firm**], [**opsys** | **util** | **appl**], and [**console**].
- **-l** *label* Specify the label component of standard format messages to be monitored. Regular expressions may be used in *label*.
- **-s** *severity* Specify the severity level of messages to be monitored. Acceptable values include **halt**, **error**, **warning**, **info**, **to fix**, and any additional severity levels defined by applications. The field is case-insensitive.
- **-m** *msg_text* Specify the text of messages to be monitored. Regular expressions may be used in *msg_text*.
- **-n** *num* Specify the minimum number of occurrences of the specified message which will cause an alert. The default is **1**. The maximum is **32**.
- **-p** *period* Specify the time period in which *num* occurrences of the specified message will cause an alert. *period* begins at the first occurrence of a message meeting the other criteria specified in the request. The default is no period, which causes an alert to be generated on *num* alone. The format of *period* is *n*

–S *time* Specify a start time, on or after which *num* occurrences of the specified message within the optional *period* will cause an alert. The format of *time* is *hhmm*, where *hh* is the hour in the range [**00**–**23**] and *mm* is the minute in the range [**00**–**59**]. If **–S** is specified and **–E** is not, the end time defaults to **2359**.

–E *time* Specify an end time, before which **num** occurrences of the specified message within the optional *period* will cause an alert. The format of *time* is *hhmm*, where *hh* is the hour in the range [**00**–**23**] and *mm* is the minute in the range [**00**–**59**]. If **–E** is specified and **–S** is not, the start time defaults to **0000**.

–e *command* Specifies a command line syntax to be executed locally when the message alerting criteria are met. *command* must be a full pathname. If it is not, an error message will be generated. *command* is executed in addition to the alert sent to the console. The alert itself is passed as the last argument to *command*.

–d *system* Specify the system to which the alert will be sent. NOTE: this option is only available when the Remote Operation Interface is running on the local system and the remote system(s). By default, an alert is sent to the local system only. However, when this option is used, the local system must be explicitly named on the command line, as well as the remote system(s), in order to receive the alert.

ident A unique identifier you assign to an alerting request when you add it to the current set of requests. The identifier must be the operand to **msgalert** when removing or resetting an alerting request, and can be used to restrict the report of the current set of requests. Where multiple identifiers are allowed, a comma-separated list can be input.

RETURN VALUE

0 Successful completion of command
10 Invalid syntax
20 Invalid argument
30 Internal system error
40 Unable to communicate with associated daemon process

FILES

/var/sadm/msgmgt/logalert (daemon)

EXAMPLE

Example 1:

```
msgalert –a –s HALT –c opsys,util ident1
```

This example specifies that all messages from the operating system or from system utilities, with the severity level of **HALT** generate alerts on the local system console.

Example 2:

```
msgalert –a –c soft –l UX:lp –s warning –d sysA ident2
```

This example specifies that all messages with the classification of **soft**, the label **UX:lp**, and the severity level of **warning** generate alerts on the console of system **sysA**. Assuming that **sysA** is a remote system, the local system console is excluded, in this case, from receiving the alert.

Example 3:

```
msgalert –a –m "this is an example" –n 3 –p 0100 ident3
```

This example specifies that alert(s) will be generated on the local console if the message, **this is an example**, occurs three or more times in any one-hour period.

SEE ALSO

lfmt(BA_OS), addsev(BA_LIB), lfmt(BA_LIB), setlabel(BA_LIB), msgrpt(AS_CMD), remop(RA_CMD).

LEVEL

Level 1.

NAME

msgrpt – log reporting facility

SYNOPSIS

msgrpt [**-o**] [**-r**] [**-c** *classification*[**,***classification*...]]
[**-m** *msg_text*] [**-l** *label*] [**-s** *severity1*[**,***severity2*...]]
[**-S** *time*] [**-E** *time*] [**-A** *date*] [**-B** *date*] [*logfile* . . .]

DESCRIPTION

msgrpt is a tool for generating reports of logged messages to standard output. It accesses the logfiles managed by the message logging and monitoring facility. The **-c** *classification*, **-m** *msg_text*, **-l** *label*, **-s** *severity*, **-S** *time*, **-E** *time*, **-A** *date*, or **-B** *date* options can be used to restrict the report to a subset of logged messages that match those criteria.

When the **-o** option, or no option, is specified, the contents of all active logfiles are displayed in chronological order. Specifying *logfile* produces a report based on only the contents of that logfile.

msgrpt can report on all currently managed or active logfiles and their associated copies—you don't need to know the specific names or locations of logfiles to generate reports. Copies are automatically included in report generation as needed to process the **-S** *time*, **-E** *time*, **-A** *date* and **-B** *date* options. In the absence of these options, **msgrpt**, by default, does not include logfile copies in its processing. In addition, if you want a report on an inactive logfile (that is, a file that is not currently the target of a message logging request) and/or its associated copies, you must specify the file(s) in the *logfile* operand.

The command options and arguments are as follows. (See **lfmt(BA_LIB)** for complete information on the standard format message components *classification*, *label*, *msg_text*, and *severity*.)

-o Omit headers from the report and display all logged messages in data format (fields are delimited by a semicolon, multiple values in a field are delimited by a comma).

-r Display the logged messages in reverse chronological order. The default is chronological order.

-S *time* Specify a start time. Only messages occurring on or after *time* will be included in the report. The format of *time* is

-B *date* Specify an end date. Only messages occurring before *date* will be included in the report. The form of *date* is *mmdd[yy]*. If *yy* is not specified, it defaults to the current year.

-c *classification* Specify the classification of messages to be reported. Acceptable values can be one from each of the keyword sets [**soft** | **hard** | **firm**], [**opsys** | **util** | **appl**], and [**console**].

-m *msg_text* Specify the text of messages to be reported. Regular expressions may be used in *msg_text*.

-l *label* Specify the label component of messages to be reported. Regular expressions may be used in *label.*

-s *severity* Specify the severity level of messages to be reported. Acceptable values include **halt**, **error**, **warning**, **info**, **to fix**, and any additional severity levels defined by an application via the **addsev** library routine. This field is case-insensitive.

logfile Specify the logfile to be used to generate the report. If a filename (versus a full pathname) is specified, **msgrpt** assumes the file is located in the default logfile directory, **/var/spool/log/current**. If no logfile is named as the operand, all active logfiles currently managed by the message logging and monitoring facility will be processed.

RETURN VALUE

The command return codes are:

0 Successful completion of command
10 Invalid syntax
20 Invalid argument
30 Internal system error

FILES

/var/spool/log/current
/var/spool/log/old

USAGE

Each message in the report is displayed in the language in which it was logged. Thus, a report may contain messages in one or more languages or character sets.

The absence of options to **msgrpt** causes all messages in all active logfiles to be selected. This usually causes more output than is desired.

EXAMPLE

Example 1:

```
msgrpt -l 'UX:*'
```

In this example, a regular expression is used as the argument to **-l** to look for matches on the label component of standard format messages. To use a regular expression as an argument, you must enclose the string in quotes, as the example illustrates. Messages in active logfiles with labels that match the regular expression **'UX:*'** will be displayed in chronological order with a header for each field. Each message in the report is presented as a unique (multiline, if needed) entry in the report, as shown below:

```
DATE/TIME,
MESSAGE
------------------------------------------------------------

01/03/90 14:25:31
UX:appl: ERROR: message for application

01/03/90 14:25:34
UX:appl: ERROR: message for application
```

Example 2:

```
msgrpt -o log1109
```

In this example, a report of the contents of **log1109** is created with no headers (data format).

```
01/03/90 14:25:31;UX:appl: ERROR: message for application\n
01/03/90 14:25:34;UX:appl: ERROR: message for application\n
```

Since **msgrpt** sends its output to standard output, this data can be piped to another process as the input. Note that multiple values in a field are delimited by a comma, and fields are delimited by a semicolon.

In the formatted report the actual date is displayed according to the user's locale setting, regardless of how the date/time stamp is stored internally.

Example 3:

```
msgrpt -r
```

In this example, the messages in all active logfiles will be displayed in reverse chronological order.

SEE ALSO

lfmt(BU_CMD), lfmt(BA_LIB), setlabel(BA_LIB), addsev(BA_LIB), msgalert(AS_CMD).

LEVEL

Level 1.

NAME

mvdir – move a directory

SYNOPSIS

`mvdir` *dirname name*

DESCRIPTION

The command `mvdir` moves directories within a file system. The argument *dirname* must be a directory; *name* must not be an existing file. If *name* is a directory, then *dirname* is moved to *name*/*dirname*, provided no such file or directory already exists. Neither name may be a sub-set of the other (`/x/y` cannot be moved to `/x/y/z`, nor vice versa).

Use of `mvdir` is restricted to a user with appropriate privileges.

USAGE

Administrator.

LEVEL

Level 1.

NAME

nice – run a command at low priority

SYNOPSIS

`nice` [–*increment*] *command*

DESCRIPTION

The command `nice` executes *command* with a lower CPU scheduling priority.

The command `nice` requires that the invoking process (generally the user's shell) be in the time-sharing scheduling class and causes *command* to be executed in the time-sharing class.

increment is a positive integer less than `{NZERO}`; if it is not given, the default is half of `{NZERO}` (rounded up).

When invoked with appropriate privileges, the `nice` command may run commands with a higher than normal priority by using a negative increment, *e.g.*, `nice --2`.

An *increment* larger than the maximum is equivalent to the maximum.

The command `nice` returns the exit status of the subject command.

ERRORS

The command `nice` will fail if the invoking process is in a scheduling class other than time-sharing.

SEE ALSO

nice(KE_OS), priocntl(RT_CMD), priocntl(RT_OS).

LEVEL

Level 1.

NAME

pkgadd – transfer software package or set to the system

SYNOPSIS

pkgadd [-d *device***] [-r** *response***] [-n] [-a** *admin***]**
[*pkginst1* **[***pkginst2***[. . .]]]**

pkgadd -s *spool* **[-d** *device***] [***pkginst1* **[***pkginst2***[. . .]]]**

DESCRIPTION

pkgadd transfers the contents of a software package or set from the distribution medium or directory to install it onto the system. A package is a collection of related files and executables that can be independently installed. On the target system, the package is either installed (by default) or spooled in a designated spool directory (if the **-s** option is used). When run without any options, **pkgadd** uses **/var/spool/pkg** as the default spool directory. The following options are available.

-d *device* Installs or copies a package/set from *device*. *device* can be: (a) the full pathname to a directory, file, or named pipe (such as **/var/tmp**); (b) the device identifiers for tape or disk devices (such as **/dev/rmt/*** or **/dev/dsk/***) (c) a device alias (such as **diskette1**); or (d) **"-"** which specifies packages in datastream format read from standard input. The default device is the installation spool directory (**/var/spool/pkg**).

For device identifiers, the device specified (either by pathname or alias), must have an entry in the device table (**/etc/device.tab**). If no entry exists in the device table, **pkgadd** will abort.

A device alias is the unique name by which a device is known. (For example, the alias for a cartridge tape drive might be **ctape1**.) The name must be limited in length to 64 characters (**DDB_MAXALIAS**) and can contain only alphanumeric characters and/or any of the following special characters: underscore (_), dollar sign (**$**

pkginst A short string used to designate a package/set. It is composed of one or two parts: *pkg* (an abbreviation for the package/set name) or, if more than one instance of that package exists, *pkg* plus *inst* (an instance identifier). (The term ''package instance'' is used loosely: it refers to all instantiations of *pkginst*, even those that do not include instance identifiers.)

The package name abbreviation (*pkg*) is the mandatory part of *pkginst*.

The second part (*inst*), which is required only if you have more than one instance of the package in question, is a suffix that identifies the instance. This suffix is either a number (preceded by a period) or any short mnemonic string you choose. If you don't assign your own instance identifier when one is required, the system assigns a numeric one by default. For example, if you have three instances of the Advanced Commands package and you don't create your own mnemonic identifiers (such as **old** and **beta**), the system adds the suffixes **.2** and **.3** to the second and third packages, automatically.

To indicate all instances of a package, specify **'***pkginst***.*'**, enclosing the command line in single quotes, as shown, to prevent the shell from interpreting the * character. Use the token **all** to refer to all packages available on the source medium.

-s *spool* Reads the package into the directory *spool* instead of installing it.

USAGE

The **-r** option can be used to indicate a directory name as well as a filename. The directory can contain numerous *response* files, each sharing the name of the package with which it should be associated. This would be used, for example, when adding multiple interactive packages with one invocation of **pkgadd**. Each package that had a request script would need a *response* file. If you create response files with the same name as the package (for example, *package1* and *package2*) then, after the **-r** option, name the directory in which these files reside.

The **-n** option will cause the installation to halt if any interaction is needed to complete it.

The **pkgadd** command checks to see if any of the files in *pkginst* are already installed on the system and, if any are, saves this fact before continuing with installation. Later, **pkgadd** won't reinstall these files on the system. If one of the package's installation scripts removes such a file, the result will be that the file will no longer be on the system when package installation completes.

The **pkgadd** command does not uncompress any files that were already compressed (that is, only those in "**.Z**" form) before being processed by **pkgmk**.

SEE ALSO

pkgask(AS_CMD), **pkgchk**(AS_CMD), **pkginfo**(AS_CMD), **pkgrm**(AS_CMD), **pkgtrans**(AS_CMD)

LEVEL

Level 1.

NAME

pkgask – stores answers to a request script

SYNOPSIS

`pkgask` [`-d` *device*] `-r` *response pkginst* [*pkginst* ...]

DESCRIPTION

The command `pkgask` allows the administrator to store answers to an interactive package (one with a request script). Invoking this command generates a *response* file that is then used as input at installation time. The use of this *response* file prevents any interaction from occurring during installation since the file already contains all of the information the package needs.

The options and arguments have the following meanings:

`-d` *device* — Runs the request script from *device*. *device* can be a directory pathname or the identifiers for tape, floppy disk or removable disk (for example, `/var/tmp`, `/dev/diskette`, and `/dev/dsk/c1d0s0`). The default device is the installation spool directory.

`-r` *response* — Identifies a file or directory, which should be created to contain the responses to interaction with the package. The name must be a full pathname. The file, or directory of files, can later be used as input to the `pkgadd` command. If *response* is a directory, `pkgask` creates a response file in that directory for each *pkginst*, using the package instance name as the response filename.

pkginst — Specifies the package instance or list of instances to be installed. The token `all` may be used to refer to all packages available on the source medium.

USAGE

The `-r` option can be used to indicate a directory name as well as a filename. The directory name is used to create numerous response files, each sharing the name of the package with which it should be associated. This would be used, for example, when adding multiple interactive packages with one invocation of `pkgadd`. Each package would need a response file. To create multiple response files with the same name as the package instance, name the directory in which the files should be created and supply multiple instance names with the `pkgask` command. When installing the packages, you will be able to identify this directory to the `pkgadd` command.

SEE ALSO

installf(AS_CMD), pkgadd(AS_CMD), pkgchk(AS_CMD), pkginfo(AS_CMD), pkgmk(AS_CMD), pkgparam(AS_CMD), pkgproto(AS_CMD), pkgrm(AS_CMD), pkgtrans(AS_CMD), removef(AS_CM).

LEVEL

Level 1.

NAME

pkgchk – check accuracy of installation

SYNOPSIS

`pkgchk` [`-l`|`-acfqv`] [`-nx`] [`-p` *path1*[`,`*path2* ...] [`-i` *file*] [*pkginst*...]

`pkgchk -d` *device* [`-l`|`v`] [`-p` *path1*[`,`*path2* ...] [`-i` *file*] [*pkginst*...]

`pkgchk -m` *pkgmap* [`-e` *envfile*] [`-l`|`-acfqv`] [`-nx`] [`-i` *file*]
[`-p` *path1*[`,`*path2* ...]]

DESCRIPTION

`pkgchk` checks the accuracy of installed files or, by use of the `-l` option, displays information about package files. The command checks the integrity of directory structures and the files. Discrepancies are reported on `stderr` along with a detailed explanation of the problem.

The first synopsis defined above is used to list or check the contents and/or attributes of objects that are currently installed on the system. Package names may be listed on the command line, or by default the entire contents of a machine will be checked.

The second synopsis is used to list or check the contents of a package which has been spooled on the specified device, but not installed. Note that attributes cannot be checked for spooled packages.

The third synopsis is used to list or check the contents and/or attributes of objects which are described in the indicated *pkgmap*.

The option definitions are:

`-l` Lists information on the selected files that make up a package. It is not compatible with the `a`, `c`, `f`, `g`, and `v` options.

`-a` Audits the file attributes only, does not check file contents. Default is to check both.

`-c` Audits the file contents only, does not check file attributes. Default is to check both.

`-f` Corrects file attributes if possible. If used with the `-x` option, it removes hidden files. When `pkgchk` is invoked with this option it creates directories, named pipes, links and special devices if they do not already exist.

`-q` Quiet mode. Does not give messages about missing files.

`-v` Verbose mode. Files are listed as processed.

`-n` Does not check volatile or editable files. This should be used for most post-installation checking.

`-x` Searches exclusive directories, looking for files which exist that are not in the installation software database or the indicated *pkgmap* file.

`-p` Only checks the accuracy of the pathname or pathnames listed. *pathname* can be one or more pathnames separated by commas (or by white space, if the list is quoted).

`-i` Reads a list of pathnames from *file* and compares this list against the installation software database or the indicated *pkgmap* file. Pathnames which are not contained in *inputfile* are not checked.

`-d` Specifies the device on which a spooled package resides. *device* can be a directory pathname or the identifiers for tape, floppy disk or removable disk (for example, `/var/tmp` or `/dev/diskette`).

`-m` Requests that the package be checked against the pkgmap file *pkgmap*.

`-e` Requests that the pkginfo file named as *envfile* be used to resolve parameters noted in the specified pkgmap file.

pkginst

Specifies the package instance or instances to be checked. The format *pkginst*`.*` can be used to check all instances of a package. The default is to display all information about all installed packages.

SEE ALSO

pkgadd(AS_CMD), pkgask(AS_CMD), pkginfo(AS_CMD), pkgrm(AS_CMD), pkgtrans(AS_CMD).

LEVEL

Level 1.

NAME

pkginfo – display software package information

SYNOPSIS

`pkginfo` [`-q`|`x`|`l`] [`-p`|`i`] [`-a` *arch*] [`-v` *version*] [`-r`]
[`-c` *category1*[`,`*category2* ...]] [*pkginst*[`,`*pkginst* ...]]

`pkginfo` [`-d` *device* [`-q`|`x`|`l`] [`-a` *arch*] [`-v` *version*]
[`-c` *category1*[`,`*category2* ...]] [*pkginst*[`,`*pkginst* ...]]

DESCRIPTION

`pkginfo` displays information about software packages which are installed on the system (with the first synopsis) or uninstalled packages which reside on a particular device or directory (with the second synopsis). Only the package name and abbreviation for pre-SVR4 packages will be included in the display.

The options for this command are:

`-q` Does not list any information, but can be used from a program to check (*i.e.*, query) whether or not a package has been installed.

`-x` Designates an extracted listing of package information. It contains the package abbreviation, package name, package architecture (if available) and package version (if available).

`-l` Designates long format, which includes all available information supplied in the package(s) `pkginfo` fields.

`-p` Designates that information should be presented only for partially installed packages.

`-i` Designates that information should be presented only for fully installed packages.

`-a` Specifies the architecture of the package as *arch*.

`-v` Specifies the version of the package as *version*. "All compatible versions" can be requested by preceding the version name with a tilde (~). Multiple white space is replaced with a single space during version comparison.

`-r` If the package is relocatable, lists the installation base for the specified package.

`-c` Selects packages to be displayed based on the category *category*. (Categories are defined in the category field of the *pkginfo* file.) If more than one category is supplied, the package must only match one of the list of categories. The match is not case specific.

pkginst Designates a package by its instance. An instance can be the package abbreviation or a specific instance (for example, `inst.1` or `inst.beta`). All instances of package can be requested by `inst.*`.

`-d` Defines a device, *device*, on which the software resides. *device* can be a directory pathname or the identifiers for tape, floppy disk, removable disk, *etc.* The special token `spool` may be used to indicate the default installation spool directory.

USAGE

Without options, `pkginfo` lists the primary category, package instance, and name of all completely installed and partially installed packages. One line per package selected is produced.

The `-p` and `-i` options cannot be used with the `-d` option.

The options `-q`, `-x`, and `-l` are mutually exclusive.

`pkginfo` cannot tell if a pre-SVR4 package is only partially installed. It assumes that all pre-SVR4 packages are fully installed.

SEE ALSO

pkgadd(AS_CMD), pkgask(AS_CMD), pkgchk(AS_CMD), pkginfo(AS_LIB), pkgrm(AS_CMD), pkgtrans(AS_CMD).

LEVEL

Level 1.

NAME

pkgmk – produce an installable package

SYNOPSIS

`pkgmk` [`-o`] [`-d` *device*] [`-r` *rootpath*] [`-b` *basdir*] [`-l` *limit*] [`-a` *arch*] [`-v` *version*] [`-p` *pstamp*] [`-f` *prototype*] [*variable=value ...*] [*pkginst*]

DESCRIPTION

The command `pkgmk` produces an installable package to be used as input to the `pkgadd` command. The package contents will be in directory structure format.

The command uses the package *prototype* file as input [see pkgproto(AS_CMD)] and creates a `pkgmap` file. The contents for each entry in the *prototype* file is copied to the appropriate output location. Information concerning the contents (checksum, file size, modification date) is computed and stored in the `pkgmap` file, along with attribute information specified in the *prototype* file.

The options and arguments have the following meanings:

`-o` Overwrites the same instance. Package instance will be overwritten if it already exists.

`-d` *device* Creates the package on *device*. *device* can be a directory pathname or the identifiers for a floppy disk or removable disk (for example, `/dev/diskette`). The default device is the installation spool directory.

`-r` *rootpath* Ignores destination paths in the *prototype* file. Instead, uses the indicated *rootpath* with the source pathname appended to locate objects on the source machine.

`-b` *basdir* Prepends the indicated *basdir* to locate relocatable objects on the source machine.

`-l` *limit* Specifies the maximum size of the output device as *limit*. By default, if the output file is a directory or a mountable device, `pkgmk` will employ the `df` command to dynamically calculate the amount of available space on the output device.

`-a` *arch* Overrides the architecture information provided in the `pkginfo` file with *arch*.

`-v` *version* Overrides version information provided in the `pkginfo` file with *version*.

`-p` *pstamp* Overrides the production stamp definition in the `pkginfo` file with *pstamp*.

`-f` *prototype* Uses the file *prototype* as input to the command. The default *prototype* filename is either of `Prototype` or `prototype`.

variable=value Places the indicated variable in the packaging environment. (*e.g.*, `BIN=$HOME/bin`).

pkginst Specifies the package by its instance. An instance can be the package abbreviation or a specific instance (for example, `inst.1`).

USAGE

Architecture information is provided on the command line with the `-a` option or in the *prototype* file. If no architecture information is supplied at all, the output of `uname -m` will be used.

Version information is provided on the command line with the `-v` option or in the *prototype* file. If no version information is supplied, a default based on the current date will be provided.

Command line definitions for both architecture and version override the *prototype* definitions.

SEE ALSO

df(BU_CMD), installf(AS_CMD), pkgparam(AS_CMD), pkgproto(AS_CMD), pkgtrans(AS_CMD), removef(AS_CMD), uname(BU_CMD).

LEVEL

Level 1.

NAME

pkgparam – display package parameter values

SYNOPSIS

`pkgparam` [`-v`] [`-d` *device*] *pkginst* [*param* ...]

`pkgparam -f` *file* [`-v`] [*param* ...]

DESCRIPTION

The command `pkgparam` displays the value associated with the parameter or parameters requested on the command line. The values are located in either the `pkginfo` file for *pkginst* or from the specific file named with the `-f` option.

One parameter value is shown per line. Only the value of a parameter is given unless the `-v` option is used. With this option, the output of the command is in this format:

parameter1='value1'
parameter2='value2'
parameter3='value3'

If no parameters are specified on the command line, values for all parameters associated with the package are shown.

The options and arguments have the following meanings:

`-v` Specifies verbose mode. Display shows name of parameter and its value.

`-d` *device* Specifies a device name. *device* can be a directory name or the identifier for tape, floppy diskette, *etc.* The special token `spool` may be used to represent the default installation spool directory. Parameter information for all packages residing in *device* are shown.

`-f` *file* Requests that the command read *file* for parameter values.

pkginst Defines a specific package instance for which parameter values should be displayed.

param Defines a specific parameter whose value should be displayed.

ERRORS

If parameter information is not available for the indicated package, the command exits with a non-zero status.

USAGE

The `-f` synopsis allows you to specify the file from which parameter values should be extracted. This file should be in the same format as a `pkginfo` file. As an example, such a file might be created during package development and used while testing software during this stage.

SEE ALSO

installf(AS_CMD), pkgmk(AS_CMD), pkgparam(AS_LIB), pkgproto(AS_CMD), pgktrans(AS_CMD), removef(AS_CMD).

LEVEL

Level 1.

NAME

pkgproto – generate prototype file entries

SYNOPSIS

`pkgproto` [`-i`] [`-c` *class*] [*path1*[=*path2*] ...]

DESCRIPTION

The command `pkgproto` scans the indicated paths and generates prototype file entries that may be used as input to the `pkgmk` command [see pkgmk(AS_CMD)].

The options and arguments have the following meanings:

`-i` Ignores symbolic links and records the paths as ftype=f (a file) versus ftype=s (symbolic link)

`-c` *class* Maps the class of all paths to *class*.

path1 Pathname where objects are located.

path2 Pathname which should be substituted on output for *path1*.

If no paths are specified on the command line, standard input is assumed to be a list of paths. If the pathname listed on the command line is a directory, the contents of the directory is searched. However, if input is read from `stdin`, a directory specified as a pathname will not be searched.

USAGE

By default, `pkgproto` creates symbolic link entries for any symbolic link encountered (ftype=s). When invoked with the `-i` option, `pkgproto` creates a file entry for symbolic links (ftype=f). The resulting prototype file entries will have to be modified to assign such file types as "v" (volatile), "e" (editable), or "x" (exclusive directory). `pkgproto` detects linked files. If multiple files are linked together, the first path encountered is considered the source of the link. The output should be saved in a file (named `Prototype` or `prototype`, for convenience) to be used as input to the `pkgmk` command.

EXAMPLES

Example 1:

```
$ pkgproto /bin=bin /usr/bin=usrbin /etc=etc
f none bin/sed=/bin/sed 0775 bin bin
f none bin/sh=/bin/sh 0755 bin daemon
f none bin/sort=/bin/sort 0755 bin bin
f none usrbin/sdb=/usr/bin/sdb 0775 bin bin
f none usrbin/shl=/usr/bin/shl 4755 bin bin
d none etc/master.d 0755 root daemon
f none etc/master.d/kernel=/etc/master.d/kernel 0644 root daemon
f none etc/rc=/etc/rc 0744 root daemon
```

Example 2:

```
$ find / –type d –print | pkgproto
d none / 755 root root
d none /bin 755 bin bin
d none /usr 755 root root
d none /usr/bin 775 bin bin
d none /etc 755 root root
d none /tmp 777 root root
```

SEE ALSO

installf(AS_CMD), pkgmk(AS_CMD), pkgparam(AS_CMD), pkgtrans(AS_CMD), removef(AS_CMD).

LEVEL

Level 1.

NAME

pkgrm – removes a package from the system

SYNOPSIS

`pkgrm [-n] [-a` *admin*`] [`*pkginst1*`[,`*pkginst2*`[, ...]]]`

`pkgrm -s` *spool* [*pkginst*]

DESCRIPTION

The command `pkgrm` will remove a previously installed or partially installed package from the system. A check is made to determine if any other packages depend on the one being removed. The action taken if a dependency exists is defined in the *admin* file.

The default state for the command is in interactive mode, meaning that prompt messages are given during processing to allow the administrator to confirm the actions being taken. Non-interactive mode can be requested with the `-n` option.

The `-s` option can be used to specify the directory from which spooled packages should be removed.

The options and arguments have the following meanings:

`-n` Non-interactive mode. If there is a need for interaction, the command will exit. Use of this option requires that at least one package instance be named upon invocation of the command.

`-a` *admin* Defines an installation admin file, *admin*, to be used in place of the (implementation dependent) default admin file.

`-s` *spool* Removes the specified package(s) from the directory *spool*.

pkginst Specifies the package to be removed. The format *pkginst* can be used to remove all instances of a package.

SEE ALSO

installf(AS_CMD), pkgadd(AS_CMD), pkgask(AS_CMD), pkgchk(AS_CMD), pkginfo(AS_CMD), pkgmk(AS_CMD), pkgparam(AS_CMD), pkgproto(AS_CMD), pkgtrans(AS_CMD), removef(AS_CMD).

LEVEL

Level 1.

NAME

pkgtrans – translate package format

SYNOPSIS

`pkgtrans [-i] [-o|n]` *device1 device2* [*pkginst1*[,*pkginst2*[,...]]]

DESCRIPTION

The command `pkgtrans` translates an installable package from one format to another. It will translate the following:

- a spool directory to a datastream,
- a datastream to a spool directory,
- a spool directory to a removable device,
- a removable device to a spool directory.

The options and arguments have the following meanings:

`-i` Copies only the `pkginfo` and `pkgmap` files.

`-o` Overwrites the same instance on the destination device (package instance will be overwritten if it already exists).

`-n` Creates a new instance if any instance of this package already exists.

device1 Indicates the source device. The package or packages on this device will be translated and placed on *device2*.

device2 Indicates the destination device. Translated packages will be placed on this device.

pkginst Specifies which package instance or instances on *device1* should be translated. The token `all` may be used to indicate all packages. *pkginst*`.*` can be used to indicate all instances of a package. If no packages are defined, a prompt shows all packages on the device and asks which to translate.

USAGE

Device specifications can be either the special node name (*e.g.* `/dev/diskette`) or the device identifier (*e.g.* `diskette1`). The device `spool` indicates the default spool directory. Source and destination devices may not be the same.

By default, `pkgtrans` will not transfer any instance of a package if any instance of that package already exists on the destination device. Use of the `-n` option will create a new instance if an instance of this package already exists. Use of the `-o` option will overwrite the same instance if it already exists. Neither of these options are useful if the destination device is a datastream.

EXAMPLE

The following example translates all packages on the floppy drive `/dev/diskette` and places the translations on `/tmp`.

```
pkgtrans /dev/diskette /tmp all
```

The next example translates packages `pkg1` and `pkg2` on `/tmp` and places their translations on the `9track1` output device.

```
pkgtrans /tmp 9track1 pkg1 pkg2
```

SEE ALSO

installf(AS_CMD), pkgadd(AS_CMD), pkgask(AS_CMD), pkginfo(AS_CMD), pkgmk(AS_CMD), pkgparam(AS_CMD), pkgproto(AS_CMD), pkgrm(AS_CMD), removef(AS_CMD).

LEVEL

Level 1.

NAME

prtconf – print system configuration

SYNOPSIS

`prtconf`

DESCRIPTION

The command `prtconf` prints the system configuration information which includes the memory and peripheral configuration. This information is displayed every time the system is initialized to multiuser mode.

EXAMPLE

To print a computer's configuration, execute:

```
$ prtconf<CR>
SYSTEM CONFIGURATION:
Memory size: 2 Megabytes
System Peripherals:
Device Name    Subdevices          Extended Subdevices
SBD
          Floppy Disk
          30 Megabyte Disk
          72 Megabyte Disk
SCSI
          SD00 ID1
                        67 Megabyte Disk ID0
                        67 Megabyte Disk ID1
                        135 Megabyte Disk ID2
                        135 Megabyte Disk ID3
NI
PORTS
CTC
```

LEVEL

Level 1.

NAME

pwck, grpck – password / group file checkers

SYNOPSIS

`pwck` [*file*]

`grpck` [*file*]

DESCRIPTION

The command `pwck` scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The default password file is `/etc/passwd`.

The command `grpck` verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file and whether the file contains more than `{NGROUPS_MAX}` entries for an individual. In addition, group entries in `/etc/group` with no login names are flagged. The default group file is `/etc/group`.

FILES

`/etc/group`
`/etc/passwd`

USAGE

Administrator.

LEVEL

Level 1.

NAME

removef – remove a file from the installation software database

SYNOPSIS

`removef` *pkginst path1* [*path2* ...]

`removef –f` *pkginst*

DESCRIPTION

The command `removef` informs the system that the user, or software, intends to remove a pathname. Output from `removef` is a list of pathnames that may be safely removed (no other packages have a dependency on them).

After all files have been processed, `removef` should be invoked with the `–f` option to indicate that the removal phase is complete.

EXAMPLE

The following shows the use of `removef` in a script:

```
echo "The following files are no longer part of
this package and are being removed."
removef $PKGINST /dev/xt[0-9][0-9][0-9] |
while read pathname
do
     echo "$pathname"
     rm –f $pathname
done
removef –f $PKGINST || exit 2
```

SEE ALSO

installf(AS_CMD), pkgadd(AS_CMD), pkgask(AS_CMD), pkgchk(AS_CMD), pkginfo(AS_CMD), pkgmk(AS_CMD), pkgparam(AS_CMD), pkgproto(AS_CMD), pkgtrans(AS_CMD), removef(AS_CMD).

LEVEL

Level 1.

NAME

restore – initiate restores of file systems, data partitions, or disks

SYNOPSIS

`restore` [`-o` *target*] [`-d` *date*] [`-m`] [`-s`] `-P` *partdev*

`restore` [`-o` *target*] [`-d` *date*] [`-n`] [`-s`] `-P` *partdev*

`restore` [`-o` *target*] [`-d` *date*] [`-m`] [`-v`] `-P` *partdev*

`restore` [`-o` *target*] [`-d` *date*] [`-n`] [`-v`] `-P` *partdev*

`restore` [`-o` *target*] [`-d` *date*] [`-m`] [`-s`] `-S` *odevice*

`restore` [`-o` *target*] [`-d` *date*] [`-n`] [`-s`] `-S` *odevice*

`restore` [`-o` *target*] [`-d` *date*] [`-m`] [`-v`] `-S` *odevice*

`restore` [`-o` *target*] [`-d` *date*] [`-n`] [`-v`] `-S` *odevice*

`restore` [`-o` *target*] [`-d` *date*] [`-m`] [`-s`] `-A` *partdev*

`restore` [`-o` *target*] [`-d` *date*] [`-n`] [`-s`] `-A` *partdev*

`restore` [`-o` *target*] [`-d` *date*] [`-m`] [`-v`] `-A` *partdev*

`restore` [`-o` *target*] [`-d` *date*] [`-n`] [`-v`] `-A` *partdev*

DESCRIPTION

The command `restore` posts requests for the restore of a data partition, a file system partition, or a disk from system-maintained archives. If the appropriate archive containing the required partition is online, the partition is restored immediately. If not, a request to restore the specified archive of the partition is posted to a restore status table. The default status table is `/etc/bkup/rsstatus.tab`. The restore request is assigned a restore job ID that can be used to monitor the progress of the restore or to cancel it. A restore request that has been posted must later be resolved by an operator [see rsoper(AS_CMD)].

`restore` may only be executed by a user with appropriate privileges.

If `restore -A` *partdev* is issued, the `fdisk` (full disk recovery) method [see fdisk(AS_CMD)] is used to repartition and repopulate disk *partdev*. *partdev* is the name of the device that refers to the entire disk.

The options and arguments have the following meanings:

`-d` *date* Restores the partition as of *date*. This may or may not be the latest archive. [See at(AU_CMD) for valid date formats.]

`-m` If the restoration cannot be carried out immediately, this option notifies the invoking user via `mail` [see mail(AS_CMD)] when the request has been completed.

`-n` Displays a list of all archived versions of the object contained in the backup history log, but does not attempt to restore the object.

`-o` *target* Instead of restoring directly to the specified object (*partdev* or *odevice*), this option restores the archive to *target*. *target* is of one of the following forms:

oname`:`*odev*
oname
`:`*odev*

where *oname* is the name of the file system to which the archive will be restored (for `-S` archives) and *odev* is the name of the partition to which the archive will be restored (for `-P` and `-A` archives).

`-s` While a `restore` operation is occurring, displays a dot (.) for each 100 blocks transferred from the destination device.

`-v` Displays the name of each object as it is restored. Only those archiving methods that restore named directories and files [see incfile(AS_CMD) and ffile(AS_CMD)] support this option.

`-A` *partdev* Initiates restore of the entire disk, *partdev*.

`-P` *partdev* Initiates restore of the data partition *partdev*.

`-S` *odevice* Initiates restore of the file system partition *odevice.*

ERRORS

The exit codes for `restore` are the following:

0 = successful completion of the task

1 = one or more parameters to `restore` are invalid.

2 = an error has occurred which caused `restore` to fail to complete all portions of its task.

FILES

`/etc/bkup/bkhist.tab`	keeps track of the location (by volume label) of all the volumes of backup archives available for use in restoring lost files as well as (optionally) the contents of each archive.
`/etc/bkup/rsstatus.tab`	lists the status of all restore requests from users.
`/etc/bkup/rsnotify.tab`	lists the email address of the operator to be notified whenever restore requests require operator intervention.

EXAMPLE

Example 1:

```
restore -m -S /usr
```

posts a request to restore the most current archived version of `/usr`. If the restore cannot be carried out immediately, notify the invoking user when the request has been completed.

Example 2:

```
restore –o /dev/rdsk/c1d0s8 –P /dev/rdsk/c1d1s2
```

posts a request that the archived data partition `/dev/rdsk/c1d1s2` be restored to the target device partition `/dev/rdsk/c1d0s8`.

Example 3:

```
restore –d "december 1, 1987" –A /dev/rdsk/c1d0s6
```

posts a request for the restore of the entire disk `/dev/rdsk/c1d0s6`. The restore should be made as of December 1, 1987.

Example 4:

```
restore –n –P /dev/rdsk/c1d0s1
```

requests the system to display the backup date and an `ls -l` listing [see ls(BU_CMD)] from the backup history log of all archived versions of the data partition `/dev/rdsk/c1d0s1`. The data partition is not restored.

SEE ALSO

at(AU_CMD), fdisk(AS_CMD), ffile(AS_CMD), getdate(BA_LIB), incfile(AS_CMD), ls(BU_CMD), mail(BU_CMD), rsnotify(AS_CMD), rsoper(AS_CMD), rsstatus(AS_CMD), urestore(AS_CMD), ursstatus(AS_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.

Optional

NAME

rsnotify – display or modify the identity of the individual in charge of restore requests

SYNOPSIS

`rsnotify` [`-a` *user*]

DESCRIPTION

The command `rsnotify`, without options, displays the name of the person who is to receive `mail` notifications [see mail(BU_CMD)] whenever restore requests require operator intervention. The display includes the date the individual was assigned.

`rsnotify` may only be executed by a user with appropriate privileges.

The option and argument have the following meaning:

`-a` *user* assigns a *user* to be the one to receive restore notifications. *user* is the user's login ID. If *user* is null, `rsnotify` sends the notices to `root`'s mail.

ERRORS

The exit codes for `rsnotify` are the following:

0 = successful completion of the task

1 = one or more parameters to `rsnotify` are invalid.

2 = an error has occurred which caused `rsnotify` to fail to complete all portions of its task.

FILES

`/etc/bkup/rsnotify.tab` provides the email address of the operator to be notified whenever restore requests require operator intervention.

`/etc/bkup/rsstatus.tab` tracks the status of all restore requests.

EXAMPLE

Example 1:

```
rsnotify -a oper3
```

assigns the individual with login ID `oper3` as the one to be notified when a restore request needing operator intervention is initiated.

SEE ALSO

restore(AS_CMD), rsstatus(AS_CMD), urestore(AS_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.
Optional

NAME

rsoper – service pending restore requests and service media insertion prompts

SYNOPSIS

`rsoper -d` *ddev* [`-j` *jobids*] [`-u` *users*] [`-m` *method*] [`-s`] [`-t`]
[`-o` *oname*[`:`*odevice*]]

`rsoper -d` *ddev* [`-j` *jobids*] [`-u` *users*] [`-m` *method*] [`-v`] [`-t`]
[`-o` *oname*[`:`*odevice*]]

`rsoper -r` *jobids*

`rsoper -c` [*jobids*]

DESCRIPTION

The command `rsoper -d` identifies media containing backup archives of file systems and data partitions, and allows an operator to complete pending `restore` and `urestore` requests [see restore(AS_CMD) and urestore(AS_CMD), respectively]. `rsoper` takes information about the archive entered on the command line and matches it against pending `restore` or `urestore` requests in the restore status table. `rsoper` then invokes the proper archiving method to read the archive and extract requested files, directories, and data partitions. As subsequent archive volumes are needed, the operator is requested to apply the appropriate archive volumes.

Depending on the information available in the `bkhist.tab` file and the media naming technique (internal or external), all options and arguments listed below may not be required. If required fields are omitted, `rsoper` issues an error message indicating the information that is needed. The command can then be reissued with the appropriate fields specified.

`rsoper` may be executed only by a user with appropriate privileges.

`rsoper -r` removes a pending restore job from the restore status table [see rsstatus(AS_CMD) and ursstatus(AS_CMD)] and notifies the requesting user that

If `mntpt=`*dir* is specified, *ddevice* is assumed to be a file system partition and *dir* is the place in the directory structure where *ddevice* will be mounted. This is only valid for `fimage` archives [see fimage(AS_CMD)]. *dlabels* is a list of volume labels either comma-separated or blank-separated. If blank separated, the entire *ddev* argument must be surrounded by quotes.

`-j` *jobids* Limits the scope of the request to the specified jobs. *jobids* is a list of restore job IDs (either comma-separated or blank-separated and surrounded by quotes).

`-m` *method* Assumes the archive on the first medium in the destination device was created by the *method* archiving operation. Valid *methods* are: `incfile`, `ffile`, `fimage`, `fdp`, `fdisk`, and any customized methods. This option is required if the backup history log is not available, does not include information about the specified archive or if `rsoper` cannot determine the format of the archive.

`-o` *oname*[`:`*odevice*]
Specifies the originating file system partition or data partition to be restored. *oname* is the name of the the originating file system. It may be null. *odevice* is the device name of the originating file system or data partition. This option is required if the backup history log is not available or does not include information about the specified archive.

`-r` *jobids* Removes the restore request for the specified jobs. *jobids* is a list of restore job IDs (either comma-separated or blank-separated and surrounded by quotes).

`-s` While a restore operation is occurring, this option displays a dot (.) for each 100 blocks transferred from the destination device.

`-t` Assumes that the media inserted in the destination device contain a table of contents for an archive. This option is required if the backup history log is not available, does not include information about the specified archive or if `rsoper` cannot determine the format of the medium.

`-u` *users* Restricts restores to restore requests issued by the specified *users*.

`-v` Displays the name of each object as it is restored. Only those archiving methods that restore named directories and files [see incfile(AS_CMD) and ffile(AS_CMD)] support this option.

ERRORS

The exit codes for `rsoper` are the following:

0 = successful completion of the task

1 = one or more parameters to `rsoper` are invalid.

2 = an error has occurred which caused `rsoper` to fail to complete all portions of its task.

If a *method* reports that no part of a restore request was completed, `rsoper` reports this fact to the user.

FILES

`/etc/bkup/bkhist.tab`	record of backup history and location of archives.
`/etc/bkup/rsstatus.tab`	status of all restore requests.
`/etc/bkup/rsnotify.tab`	email address of operator to be notified when restore requests require operator intervention.

EXAMPLE

Example 1:

```
rsoper –d /dev/tape/c4d0s2
```

asks the restore service to read the archive volume that has been inserted into the device `/dev/tape/c4d0s2`. The service will attempt to resolve any restore requests that can be satisfied by the archive volume.

Example 2:

The following example assumes that the backup history table contains a record of backups performed and that the restore status table has a record of the restore requests. With these assumptions:

```
rsoper –d /dev/ctape:density=1600:USRLBL1 –v –u clerk1
```

instructs the restore service to perform only those pending restore requests from the `rsstatus.tab` issued by user `clerk1`. The restorals are to be done from the cartridge tape labeled `USRLBL1`, with a density of 1600 bps. The restore service will display on the operator terminal the names of the files and directories as they are successfully restored.

Example 3:

The following example assumes that the backup history table no longer contains a log of the requested backup operations. With that assumption:

```
rsoper –d \
/dev/diskette2:blk_fac=2400:arc.dec79.a,arc.dec79.b,\
arc.dec79.c –m incfile –o /usr2
```

instructs the restore service to perform a restore of the `/usr2` file system using the incremental restore method. The `/usr2` file system is to be restored from archived diskettes with a blocking factor of 2400. The diskettes containing the archive are labeled `arc.dec79.a`, `arc.dec79.b`, and `arc.dec79.c`.

Example 4:

```
rsoper –c rest-737b
```

cancels the restore request with job ID `rest-737b`.

SEE ALSO

fimage(AS_CMD), ffile(AS_CMD), fdp(AS_CMD), fdisk(AS_CMD), getdate(BA_LIB), incfile(AS_CMD), mail(BU_CMD), restore(AS_CMD), rsnotify(AS_CMD), rsstatus(AS_CMD), urestore(AS_CMD), ursstatus(AS_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.

Optional

NAME

rsstatus – report the status of posted restore requests

SYNOPSIS

`rsstatus` [`-h`][`-d` *ddev*][`-j` *jobids*][`-u` *users*][`-f` *c*]

DESCRIPTION

With no options, the command `rsstatus` reports the status of all pending restore requests that are posted in the restore status table.

`rsstatus` may only be executed by a user with appropriate privileges.

Note: volume labels marked with an asterisk(*) in the output of this command are table of contents volumes.

The options and arguments have the following meanings:

`-d` *ddev* Restricts the report to pending restore jobs that could be satisfied by the specified device or volumes. *ddev* describes the device or volumes used to select requests to be restored. *ddev* is of the form:

[*dtype*][:*dlabels*]

dtype or *dlabels* may be `null`, but not both.

dtype is a device type, (such as diskette, ctape, 9track). If specified, restrict the report to posted requests that could be satisfied by media of the specified type.

dlabels is a list of volumes corresponding to a *volumename* on the `labelit` command [see `labelit` in volcopy(AS_CMD)]. *dlabels* may either be comma-separated or blank-separated and surrounded by quotes. If specified, restrict the report to posted requests that could be satisfied by an archive residing on the specified volumes.

`-f` *c* Suppresses field wrap and specifies an output field separator to be used. *c* is the character that will appear as the field separator on the display output. For clarity of output, do not use a separator character that is likely to occur in a field. For example, do not use the colon as a field separator character if the display will contain dates that use a colon to separate hours from minutes.

`-h` Suppresses headers for the report.

`-j` *jobids* Restricts the report to the specified jobs. *jobids* is a list of restore job IDs (either comma-separated or blank-separated and surrounded by quotes) [see restore(AS_CMD)].

`-u` *users* Restricts the report to requests submitted by the specified *users* (either comma-separated or blank-separated and surrounded by quotes).

ERRORS

The exit codes for `rsstatus` are the following:

0 = successful completion of the task

1 = one or more parameters to `rsstatus` are invalid.

2 = an error has occurred which caused `rsstatus` to fail to complete all portions of its task.

FILES

`/etc/bkup/rsstatus.tab` tracks the status of all restore requests from users

EXAMPLE

Example 1:

```
rsstatus –d diskette
```

reports the status of those posted restore requests that can be satisfied by inserting diskettes into a diskette drive.

Example 2:

```
rsstatus –j rest-354a,rest-429b
```

reports the status only of the two posted restore requests with the specified job IDs.

SEE ALSO

restore(AS_CMD), urestore(AS_CMD), ursstatus(AS_CMD), volcopy(AS_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.

Optional

NAME

runacct – run daily accounting

SYNOPSIS

runacct [*mmdd* **[***state***]]**

DESCRIPTION

runacct is the main daily accounting shell procedure. It is normally initiated via **cron**. **runacct** processes connect, fee, disk, and process accounting files. It also prepares summary files for **prdaily** or billing purposes. **runacct** is distributed only to source code licensees.

runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into *active*. When an error is detected, a message is written to **/dev/console**, mail [see **mail**(BU_CMD)] is sent to **root** and **adm**, and **runacct** terminates. **runacct** uses a series of lock files to protect against re-invocation. The files **lock** and **lock1** are used to prevent simultaneous invocation, and **lastdate** is used to prevent more than one invocation per day.

runacct breaks its processing into separate, restartable *states* using *statefile* to remember the last *state* completed. It accomplishes this by writing the *state* name into *statefile*. **runacct** then looks in *statefile* to see what it has done and to determine what to process next. *states* are executed in the following order:

SETUP Move active accounting files into working files.

WTMPFIX Verify integrity of **wtmp** file, correcting date changes if necessary.

CONNECT Produce connect session records in **tacct.h** format.

PROCESS Convert process accounting records into **tacct.h** format.

MERGE Merge the connect and process accounting records.

FEES Convert output of **chargefee** into **tacct.h** format and merge with connect and process accounting records.

DISK Merge disk accounting records with connect, process, and fee accounting records.

MERGETACCT

EXAMPLES

To start **runacct**:

```
nohup runacct 2 > /var/adm/acct/nite/fd2log &
```

To restart **runacct**:

```
nohup runacct 0601 2 >> /var/adm/acct/nite/fd2log &
```

To restart **runacct** at a specific *state*:

```
nohup runacct 0601 MERGE 2 >> /var/adm/acct/nite/fd2log &
```

SEE ALSO

acct (AS_CMD), **acctcms** (AS_CMD), **acctcom** (AS_CMD), **acctcon** (AS_CMD), **acctmerg** (AS_CMD), **cron** (AU_CMD), **fwtmp** (AS_CMD), **mail** (BU_CMD),

LEVEL

Level 2.

NOTICES

Normally it is not a good idea to restart **runacct** in the **SETUP** *state*. Run **SETUP** manually and restart via:

runacct *mmdd* **WTMPFIX**

If **runacct** failed in the **PROCESS** *state*, remove the last **ptacct** file because it will not be complete.

NAME

sa1, sa2, sadc – system activity report package

SYNOPSIS

`sadc` [*t n*] [*ofile*]

`sa1` [*t n*]

`sa2` [*options*] [`-s` *time*] [`-e` *time*] [`-i` *sec*]

DESCRIPTION

System activity data can be accessed at the special request of a user [see sar(AS_CMD)] and automatically on a routine basis as described here. The operating system contains several counters that are incremented as various system actions occur. These include counters for CPU utilization, buffer usage, disk and tape I/O activity, TTY device activity, switching and system-calls, file-access, queue activity, and interprocess communications, paging, and remote file sharing.

The commands `sa1`, `sa2`, and `sadc`, are used to sample, save, and process this data, respectively.

The command `sa1`, a variant of `sadc`, is used to collect and store data in the binary file `/var/adm/sa/sa`*dd* where *dd* is the current day. The options *t* and *n* cause records to be written *n* times at an interval of *t* seconds (once if the options are omitted).

The command `sa2`, a variant of `sar`, writes a daily report in the file `/var/adm/sa/sar`*dd*. The options are explained in sar(AS_CMD).

The command `sadc`, the data collector, samples system data *n* times with an interval of *t* seconds between samples. If *t* and *n* are omitted, a special record is written. This facility is typically used at system boot time to mark the time at which the counters restart from zero.

FILES

`/var/adm/sa/sa`*dd*	daily data file
`/var/adm/sa/sar`*dd*	daily report file

USAGE

Administrator.

SEE ALSO

sar(AS_CMD).

LEVEL

Level 2: September 30, 1989.

NAME

sacadm – service access controller administration

SYNOPSIS

`sacadm -a -p` *pmtag* `-t` *type* `-c` *cmd* `-v` *ver* [`-f` *flags*] [`-n` *count*] [`-y` *comment*] [`-z` *script*]

`sacadm -r -p` *pmtag*

`sacadm -s -p` *pmtag*

`sacadm -k -p` *pmtag*

`sacadm -e -p` *pmtag*

`sacadm -d -p` *pmtag*

`sacadm -l` [`-p` *pmtag*]

`sacadm -l` [`-t` *type*]

`sacadm -L` [`-p` *pmtag*]

`sacadm -L` [`-t` *type*]

`sacadm -g -p` *pmtag* [`-z` *script*]

`sacadm -G` [`-z` *script*]

`sacadm -x` [`-p` *pmtag*]

DESCRIPTION

The command `sacadm` handles the administration of the top level of the Service Access Facility hierarchy. The functions provided by this command are:

Add / Remove a port monitor

Start / Stop a port monitor

Enable / Disable a port monitor

Miscellaneous requests

Normally, `sacadm` may only be executed by the system administrator or by administrative command scripts. However, requests about the status of port monitors, or displaying the per-port monitor and per-system configuration scripts, may be executed by any user on the system.

The options and arguments have the following meanings:

`-a` This option is used to add an entry to the system's list of port monitors.

`-c` *cmd* This option is used to specify the command string, *cmd*, that is to be executed to start the port monitor.

`-d` This option is used to disable a port monitor.

`-e` This option is used to enable a port monitor.

`-f` *flags* This option is used to specify which flags should be set for the port monitor entry. If not specified, no flags are set. The flags must be one or both of `d` or `x` and have the following meanings:

`d` — When started, the port monitor should be placed into the disabled state. (Default: the port monitor starts in the enabled state).

`x` — The port monitor should not be started. (Default: the port monitor should be started).

A `-f` with no following argument is illegal.

`-g` This option is used to indicate that a per-port monitor configuration script should be replaced or displayed.

`-G` This option is used to indicate that the per-system configuration script should be replaced or displayed.

`-k` This option is used to stop a port monitor.

`-l` This option is used to display port monitor information.

`-L` This option is used to display port monitor information in a condensed format.

`-n` *count* This option is used to specify the restart count. If not specified, a count of 0 will be used (*i.e.* do not restart the port monitor if it fails).

`-p` *pmtag* This option is used to specify the tag associated with the port monitor.

`-r` This option is used to remove a port monitor entry from the system's list of port monitors.

`-s` This option is used to start a port monitor.

`-t` *type* This option is used to specify the type of the port monitor.

`-v` *ver* This option is used to specify the version number of the port monitor's database file.

`-x` This option is used to indicate that database information should be read.

`-y` *comment* This option is used to specify a *comment* that should be associated with the port monitor entry.

`-z` *script* This option is used to specify the name of the file that contains a configuration script. This file will be copied to the appropriate place.

When adding a port monitor, `sacadm` will create the supporting directory structure in `/etc/saf/sac.d` and will add an entry for the port monitor in the Service Access Controller (SAC) administrative file. The new port monitor will be started if that action was specified.

When removing a port monitor, `sacadm` will remove the port monitor's entry from SAC's administrative file. If the removed port monitor is not running, then no further action is taken. If the removed port monitor is running, the Service Access Controller will send it `SIGTERM` [see signal(BA_ENV)] to indicate that it should shut down.

A request to start a port monitor will cause the SAC to start the port monitor specified by *pmtag*. A request to stop a port monitor will cause the SAC to send `SIGTERM` to the port monitor specified by *pmtag*.

A request to enable a port monitor will cause the SAC to send an enable message to the port monitor specified by *pmtag*. A request to disable a port monitor will cause the SAC to send a disable message to the port monitor specified by *pmtag*.

When requesting status information or updating configuration scripts, the options may be specified in the following combinations. The `-l` by itself will cause a listing of all port monitors on the system. A `-l` in combination with a `-p` will cause a listing of the specified port monitor only. A `-l` in combination with a `-t` will cause a listing of all port monitors of the specified type. Other combinations with `-l` are implementation dependent, and may be invalid.

The `-L` option is identical to the `-l` option specified above except that the output appears in a condensed format.

The `-g` option in combination with the `-p` option will cause the configuration script for the specified port monitor to be displayed. The `-g` option in combination with the `-p` option and `-z` option will cause the specified file to be installed as the per-port monitor configuration script for the designated port monitor. Other combinations with `-g` are implementation dependent, and may be invalid.

The `-G` option by itself will cause the per-system configuration script to be displayed. The `-G` option in combination with the `-z` option will cause the specified file to be installed as the per-system configuration script. Other combinations with `-G` are implementation dependent, and may be invalid.

The `-x` option by itself specifies that the SAC should read its administrative file. The `-x` option in combination with the `-p` option specifies that the designated port monitor should read its administrative file.

RETURN VALUE

If successful, this command will exit with a status of 0. If it fails for any reason, it will exit with a nonzero status. Options that request information will result in that information being written on the standard output. In the condensed format, port monitor information will appear in colon-separated columns and the comment field will be preceded by the character #. In the regular format, a header identifying the columns will be output and the port monitor information will be aligned under the appropriate heading.

EXAMPLE

To add a port monitor whose tag is `npack`, whose type is `listen`, which will restart 3 times before failing, whose administrative command is `pmspec`, and whose configuration script is in the file `script`, use:

```
sacadm -a -p npack -t listen -c "/usr/lib/saf/listen npack"\
     -v `pmspec -V` -n 3 -z script
```

To remove a port monitor whose tag is `foo` use:

```
sacadm -r -p foo
```

To start the port monitor whose tag is `foo` use:

```
sacadm –s –p foo
```

To stop the port monitor whose tag is `foo` use:

```
sacadm –k –p foo
```

To enable the port monitor whose tag is `foo` use:

```
sacadm –e –p foo
```

To disable the port monitor whose tag is `foo` use:

```
sacadm –d –p foo
```

To list the status information about all port monitors use:

```
sacadm –l
```

To list the status information about the port monitor whose tag is `foo` use:

```
sacadm –l –p foo
```

To list the same information as above in condensed format use:

```
sacadm –L –p foo
```

To list the status information about all port monitors whose type is `listen` use:

```
sacadm –l –t listen
```

To change the configuration script associated with the port monitor whose tag is `foo` to be what is in the file `file.config` use:

```
sacadm –g –p foo –z /var/tmp/file.config
```

LEVEL

Level 1.

NAME

sadp – disk access profiler

SYNOPSIS

`sadp` [`-th`] [`-d` *device* [*-drive*]] *s* [*n*]

DESCRIPTION

The command `sadp` reports disk access location and seek distance, in tabular or histogram form. It samples disk activity once every second during an interval of *s* seconds. This is repeated *n* times if *n* is specified.

The argument *drive* specifies the disk drives and it may be:

a drive number in the range supported by *device*,

or

two numbers separated by a minus (indicating an inclusive range),

or

a list of drive numbers separated by commas.

The `-d` option may be omitted, if the system has only one *device* type.

The `-t` option (default) causes the data to be reported in tabular form. The `-h` option produces a histogram.

USAGE

Administrator.

LEVEL

Level 2: September 30, 1989.

Optional.

NAME

sar – system activity reporter

SYNOPSIS

`sar` [*options*] [`-o` *file*] *t* [*n*]

`sar` [*options*] [`-s` *time*] [`-e` *time*] [`-i` *sec*] [`-f` *file*]

DESCRIPTION

In the first synopsis above, the `sar` command samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the `-o` option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second form, with no sampling interval specified, `sar` extracts data from a previously recorded *file*, either the one specified by the `-f` option or, by default, the standard system activity daily data file `/var/adm/sa/sa`*dd* for the current day *dd*. The starting and ending times of the report can be bounded via the `-s` and `-e` *time* arguments of the form *hh*[:*mm*[:*ss*]]. The `-i` option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by the following options:

`-u` Report CPU utilization (the default):
`%usr`, `%sys`, `%wio`, `%idle` – portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.

`-b` Report buffer activity:
`bread/s`, `bwrit/s` – transfers per second of data between system buffers and disk or other block devices;
`lread/s`, `lwrit/s` – accesses of system buffers;
`%rcache`, `%wcache` – cache hit ratios, *i.e.*, 1 – bread/lread;
`pread/s`, `pwrit/s` – transfers via raw (physical) device mechanism.

`-d` Report activity for each block device, *e.g.*, disk or tape drive. When data is displayed, the device specification `dsk-` is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent. The activity data reported is:
`%busy`, `avque` – portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
`r+w/s`, `blks/s` – number of data transfers from or to device, number of bytes transferred in 512-byte units;
`avwait`, `avserv` – average time in milliseconds. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

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

`-c` Report system calls:
`scall/s` – system calls of all types;
`sread/s`, `swrit/s`, `fork/s`, `exec/s` – specific system calls;
`rchar/s`, `wchar/s` – characters transferred by read and write system calls.

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

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

`-q` Report average queue length while occupied, and percent of time occupied:
`runq-sz`, `%runocc` – run queue of processes in memory and runnable;
`swpq-sz`, `%swpocc` – swap queue of processes swapped out but ready to run.

`-v` Report status of process, i-node, file, and record lock tables:
`proc-sz`, `inod-sz`, `file-sz`, `lock-sz` – entries/size for each table, evaluated once at sampling point;
`ov` – overflows that occur between sampling points for each table.

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

`-A` Report all data (all options effective).

EXAMPLE

To see today's CPU activity so far:

```
sar
```

To watch CPU activity evolve for 10 minutes and save data:

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

To later review disk and tape activity from that period:

```
sar -d -f temp
```

FILES

`/var/adm/sa/sa`*dd* daily data file, where *dd* are digits representing the day of the month.

USAGE

Administrator.

SEE ALSO

sa1(AS_CMD).

LEVEL

Level 2: September 30, 1989.

NAME

setmnt – establish mount table

SYNOPSIS

`setmnt`

DESCRIPTION

The command `setmnt` creates the `/etc/mnttab` table, which is needed for both the `mount` and `umount` [see mount(AS_CMD)] commands. The command `setmnt` reads standard input and creates a `mnttab` entry for each line. Input lines have the format:

filesys node

where *filesys* is the name of the *special file* of the file system and *node* is the root name of that file system. Thus, *filesys* and *node* become the first two strings in the `/etc/mnttab` entry.

FILES

`/etc/mnttab`

USAGE

Administrator.

SEE ALSO

mount(AS_CMD).

FUTURE DIRECTIONS

In the future, the `mnttab` file will be replaced by a kernel table access mechanism. This will make `setmnt` obsolete.

LEVEL

Level 2: September 30, 1989.

NAME

setuname – changes machine information

SYNOPSIS

`setuname -s` *name* [`-n` *node*] [`-t`]

`setuname` [`-s` *name*] `-n` *node* [`-t`]

DESCRIPTION

The command `setuname` changes the parameter value for the system name and node name. Each parameter can be changed using `setuname` and the appropriate option. Both values may be changed at the same time. At least one value option must be given.

The options and arguments have the following meanings:

`-s` *name* Changes the system name. *name* specifies new system name and may consist of alphanumeric characters and the special characters dash, underscore, and dollar sign (which has significance to the shell, and should therefore be escaped).

`-n` *node* Changes the node name. *node* specifies the new network node name and may consist of alphanumeric characters and the special characters dash, underscore, and dollar sign.

`-t` Temporary change. No attempt will be made to preserve the change across reboots.

Either or both the `-s` and `-n` options must be given when invoking `setuname`.

The maximum size of the elements will be `SYS_NMNL` (defined in `limits.h`). The command will issue a warning message if the name entered is incompatible with the system requirements.

USAGE

`setuname` attempts to change the parameter values in two places: the running kernel and, as necessary per implementation, to cross system reboots. A temporary change changes only the running kernel.

SEE ALSO

uname(BA_OS), uname(BU_CMD).

FUTURE DIRECTIONS

Programmatic interfaces have been assured a consistent value for the string that reflects the operating system implementation name. By convention this should reflect the fact that this implementation is UNIX System V. Since, by definition, it is inappropriate to change the name of the operating system implementation without recompiling the kernel, the `-s` option will be eliminated in a future release.

LEVEL

Level 1.

The setuname `-s` *name* option has been moved to Level 2, July 1992.

NAME

srchtxt – display contents of, or search for a text string in, message data bases

SYNOPSIS

`srchtxt [-s] [-l` *locale*`] [-m` *msgfile*`[,`*msgfile* ...`]] [`*text*`]`

DESCRIPTION

The command `srchtxt` is used to display all the text strings in message data bases, or to search for a text string in message data bases [see mkmsgs(AS_CMD)]. These data bases are files in the directory `/usr/lib/locale/`*locale*`/LC_MESSAGES` [see setlocale(BA_OS)], unless a file name given with the `-m` option contains a slash (`/`). The directory *locale* can be viewed as the language in which the text strings are written. If the `-l` option is not specified, the files accessed will be determined by the value of the environment variable `LC_MESSAGES`. If `LC_MESSAGES` is not set, the files accessed will be determined by the value of the environment variable `LANG`. If `LANG` is not set, the files accessed will be in the directory `/usr/lib/locale/C/LC_MESSAGES`, which contains U. S. English strings.

If no *text* argument is present, then all the text strings in the files accessed will be displayed.

The options and arguments have the following meanings:

`-s` suppress printing of the message sequence numbers of the messages being displayed

`-l` *locale* access files in the directory `/usr/lib/locale/`*locale*`/LC_MESSAGES`

`-m` *msgfile* access file(s) specified by one or more *msgfile*s. If *msgfile* contains a slash (`/`), then *msgfile* is interpreted as a pathname; otherwise, it will be assumed to be in the directory determined as described above. To specify more than one *msgfile*, the file names are separated using commas.

text search for the text string specified by *text* and display each one that matches. *text* can take the form of a regular expression [see ed(BU_CMD)].

If the `-s` option is not specified, the displayed text is prefixed by message sequence numbers. The message sequence numbers are enclosed in angle brackets: <*msgfile*`:`*msgnum*>.

msgfile name of the file where the displayed text occurred

msgnum sequence number in *msgfile* where the displayed text occurred

This display is in the format used by the command `gettxt` [see gettxt(BU_CMD)] and the routine `gettxt()` [see gettxt(BA_LIB)].

EXAMPLE

The following examples show uses of `srchtxt`.

Example 1:

If message files have been installed in a locale named `french` by using `mkmsgs`, then you could display the entire set of text strings in the `french` locale (`/usr/lib/locale/french/LC_MESSAGES/*`) by typing:

```
srchtxt –l french
```

Example 2:

If a set of error messages associated with the operating system had been installed in the file `UX`, then you could search for the pattern `open.*file` in the file `UX` in the `french` locale (`/usr/lib/locale/french/LC_MESSAGES/UX`), using the value of the `LANG` environment variable to determine the locale to be searched, by typing:

```
LANG=french; export LANG
srchtxt –m UX "open.*file"
```

If `/usr/lib/locale/french/LC_MESSAGES/UX` contained the following strings:

```
I/O error\n
file %s:\texists\n
Too many open files\n
Error: %s open file error\n
  .
  .
  .
```

then two strings would be displayed:

```
<UX:3>Too many open files\n
<UX:4>Error: %s open file error\n
```

Example 3:

If a set of error messages associated with the operating system had been installed in the file `UX` and a set of error messages associated with a data base product had been installed in the file `ingress`, both in the `german` locale, then you could search for the pattern `close.*file` in both the files `UX` and `ingress` in the `german` locale by typing:

```
srchtxt –l german –m UX,ingress "close.*file"
```

FILES

`/usr/lib/locale/C/LC_MESSAGES/*`	default message files created by `mkmsgs`
`/usr/lib/locale/`*locale*`/LC_MESSAGES/*`	message files created by `mkmsgs`

SEE ALSO

ed(BU_CMD), gettxt(BA_LIB), gettxt(BU_CMD), mkmsgs(AS_CMD), setlocale(BA_OS).

LEVEL

Level 1.

NAME

sync – flush system buffers

SYNOPSIS

sync

DESCRIPTION

The command sync executes the sync() system routine. If the system is to be stopped, sync must be executed to ensure file system integrity. It flushes all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point are saved.

USAGE

Administrator.

SEE ALSO

sync(BA_OS).

LEVEL

Level 1.

NAME

sysdef – system definition

SYNOPSIS

`sysdef` [*opsys* [*master*]]

DESCRIPTION

The command `sysdef` analyzes the named operating system file *opsys* (or the default file if none is specified) and extracts configuration information. The *master* file contains hardware and software specifications used to create the operating system file *opsys*. (The default master file is used if one is not specified.) The *master* file may include hardware configuration as well as software devices and tunable parameters. The format of the *master* file is implementation defined.

USAGE

Administrator.

LEVEL

Level 1.

NAME

timex – time a command; report process data and system activity

SYNOPSIS

`timex [-pos]` *command*

`timex -p[fhkmrt]` *command*

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of `timex` is written on standard error.

The options have the following meanings:

`-p` List process accounting records for *command* and all its children. Suboptions `f`, `h`, `k`, `m`, `r`, and `t` modify the data items reported, as defined in acctcom(AS_CMD). The number of blocks read or written and the number of characters transferred are always reported.

`-o` Report the total number of blocks read or written and total characters transferred by *command* and all its children.

`-s` Report total system activity (not just activity due to *command*) that occurred during the execution interval of *command*. All the data items listed in sar(AS_CMD) are reported.

SEE ALSO

acctcom(AS_CMD), sar(AS_CMD).

LEVEL

Level 2: September 30, 1989.

NAME

urestore – request restore of files and directories

SYNOPSIS

`urestore` [`-o` *target*] [`-d` *date*] [`-m`] [`-s`] `-F` *file* ...

`urestore` [`-o` *target*] [`-d` *date*] [`-n`] [`-s`] `-F` *file* ...

`urestore` [`-o` *target*] [`-d` *date*] [`-m`] [`-v`] `-F` *file* ...

`urestore` [`-o` *target*] [`-d` *date*] [`-n`] [`-v`] `-F` *file* ...

`urestore` [`-o` *target*] [`-d` *date*] [`-m`] [`-s`] `-D` *dir* ...

`urestore` [`-o` *target*] [`-d` *date*] [`-n`] [`-s`] `-D` *dir* ...

`urestore` [`-o` *target*] [`-d` *date*] [`-m`] [`-v`] `-D` *dir* ...

`urestore` [`-o` *target*] [`-d` *date*] [`-n`] [`-v`] `-D` *dir* ...

`urestore -c` *jobid*

DESCRIPTION

The command `urestore` posts requests for the restore of files or directories from system-maintained archives. If the appropriate archive containing the requested files or directories is on-line, the files or directories are restored immediately. If not, a request to restore the specified files or directories is posted to a restore status table. A restore request that has been posted must later be resolved by an operator [see rsoper(AS_CMD)]. Each file or directory to be restored is assigned a restore *jobid* that can be used to monitor the progress of the restore [see ursstatus(AS_CMD)] or to cancel it.

The user must have write permission for the current directory and any subdirectories to be traversed in storing the restored files or directories. Requests for restores may be made only by the user who owned the files or directories at the time the archive containing the files or directories was made, or by a user with appropriate privileges.

`urestore -c` cancels a previously issued restore request.

The options and arguments have the following meanings:

`-d` *date* Restores the file system or directory as of *date*. (This may or may not be the latest archive.) [See at(AU_CMD) for valid date formats.]

`-m` If the restore cannot be carried out immediately, this option notifies the invoking user via `mail` [see mail(BU_CMD)] when the request has been completed.

`-n` Displays a list of all archived versions of the file system or directory contained in the backup history log but does not attempt to restore the file system or directory.

`-o` *target* Instead of restoring directly to the specified file or directory, this option replaces the file *target* with the archive of the specified file or directory.

`-s` While a restore operation is occurring, displays a dot (.) for each 100 blocks transferred from the destination device.

`-v` Displays the name of each object as it is restored. Only those archiving methods that restore named directories and files [see incfile(AS_CMD) and ffile(AS_CMD)] support this option.

`-D` *dir* Initiates restore of directories.

`-F` *file* Initiates restore of files.

ERRORS

The exit codes for `urestore` are the following:

`0 =` successful completion of the task

`1 =` one or more parameters to `urestore` are invalid.

`2 =` an error has occurred which caused `urestore` to fail to complete all portions of its task.

FILES

`/etc/bkup/bkhist.tab`	keeps track of the location (by volume label) of all the volumes of backup archives available for use in restoring lost files as well as (optionally) the contents of each archive.
`/etc/bkup/rsstatus.tab`	lists the status of all restore requests from users.
`/etc/bkup/rsnotify.tab`	lists the email address of the operator to be notified whenever restore requests require operator intervention.

EXAMPLE

Example 1:

```
urestore –m –F bigfile
```

posts a request to restore the most current archived version of the file `bigfile`. If the restore cannot be carried out immediately, it notifies the invoking user when the request has been completed.

Example 2:

```
urestore –c rest-256a,rest-256b
```

cancels restore requests with job ID numbers `rest-256a` and `rest-256b`.

Example 3:

```
urestore –o /testfiles/myfile.b –F /testfiles/myfile.a
```

posts a request for the archived file `/testfiles/myfile.a` to be restored as `/testfiles/myfile.b`.

Example 4:

```
urestore –d "december 1, 1987" –D /user1
```

posts a request for the archived directory structure `/user1`, with all its files and subdirectories, to be restored as of December 1, 1987. If the restore is done immediately from an on-line archive, the name of each file will be displayed on standard output while the restore is occurring.

Example 5:

```
urestore –n –D /pr3/reports
```

requests the system to display the backup dates and an `ls -l` listing [see ls(BU_CMD)] from the backup history log of all archived versions of the directory `/pr3/reports`. A restore of the directory is not performed.

SEE ALSO

ffile(AS_CMD), getdate(BA_LIB), incfile(AS_CMD), ls(BU_CMD), mail(BU_CMD), restore(AS_CMD), ursstatus(AS_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.

Optional

NAME

ursstatus – report the status of posted user file and directory restore requests

SYNOPSIS

`ursstatus` [`-f` *c*] [`-h`] [`-j` *jobids*]

DESCRIPTION

With no options, `ursstatus` reports the status of all pending restore requests issued by the user that are posted in the restore status table.

The options and arguments have the following meanings:

`-f` *c* Suppresses field wrap and specifies an output field separator to be used. *c* is the character that will appear as the field separator on the display output.

`-h` Suppresses headers for the report.

`-j` *jobids* Restricts the report to the specified jobs. *jobids* is a list of restore job IDs (either comma-separated or blank-separated and surrounded by quotes).

ERRORS

The exit codes for `ursstatus` are the following:

0 = successful completion of the task

1 = one or more parameters to `ursstatus` are invalid.

2 = an error has occurred which caused `ursstatus` to fail to complete all portions of its task.

FILES

`/etc/bkup/rsstatus.tab` tracks the status of all restore requests from users

EXAMPLE

```
ursstatus -j rest-354a,rest-429b
```

reports the status only of the two posted restore requests with the specified job IDs.

SEE ALSO

restore(AS_CMD), rsstatus(AS_CMD), urestore(AS_CMD).

FUTURE DIRECTIONS

This command will be modified in the future in a way that provides compliance with any eventual POSIX and X/Open standards and an orderly migration from current practice.

LEVEL

Level 2, April 1991.

Optional

NAME

useradd – add a new user login on the system

SYNOPSIS

useradd [-u *uid* [-o] [-i]] [-g *group*] [-G *group*[[,*group*] . . .]] [-d *dir*] [-s *shell*] [-c *comment*] [-m [-k *skel_dir*] -f *inactive*] [-e *expire*] [-h *level* [-h *level* [. . .]]] [-v *def_level*] [-w *hd_level*] [-a *event*[, . . .]] *login*

DESCRIPTION

The useradd command adds a new user entry to the Identification and Authentication (I&A) data files. It also creates supplementary group memberships for the user and creates the home directory for the user if requested. The new login remains locked until the passwd command [see passwd(AU_CMD)] is executed to set the password.

The following options are available:

-u *uid* Specifies the UID of the new user. It must be a non-negative decimal integer below {MAXUID}. It defaults to the next available UID greater than 99.

-o This option allows a UID to be duplicated (non-unique). Because the security of the system in general, and the integrity of the audit trail and accounting information in particular, depends on every UID being uniquely associated with a specific individual, never use this feature to assign more than one user the same UID.

-i Allows a UID currently being aged to be used.

-g *group* Specifies an existing group's integer ID or character string name. It defines the new user's primary group membership and defaults to the default group defined for the system.

-G *group* Specifies a set of existing groups by integer ID or by character string name. It defines the new user's supplementary group membership. Duplicates are ignored. No more than NGROUPS_MAX groups should be specified.

-d *dir* Specifies the home directory of the new user. This field is limited to {PATH_MAX} characters. *dir* is a pathname, the default value of which is *HOMEDIR/login*, where *HOMEDIR* is the base directory for new login home directories and *login* is the new login.

-s *shell* Specifies the full pathname of the program that will be used as the user's shell on login. This field is limited to {PATH_MAX} characters. It defaults to sh [see sh(BU_CMD)]. The *shell* must be a valid executable file.

-c *comment* Specifies any text string. It is generally a short description of the login, and is currently used as the field for the user's full name. It is limited to 128 printable characters and should not include colons (:).

`-m` Creates the new user's home directory if it doesn't already exist. If the directory already exists, the user being added must have read, write, and execute permissions for the directory.

`-k` *skel_dir* Specifies a directory that contains skeleton information (such as a `.profile` file) to copy into the new user's home directory. The directory must exist. The system provides a ''skel'' directory that can be used for this purpose.

`-f` *inactive* The maximum number of days allowed between uses of a login before that login is declared invalid. Normal values are positive integers. A value of `-1` overrides the value of *inactive* previously set with `-f`.

`-e` *expire* The date on which a login can no longer be used; after this date, no user will be able to access this login. (This option is useful for creating temporary logins.) You may type the value of the argument *expire* (which is a date) in any format you like (except a Julian date). For example, you may enter `10/6/90` or `October 6, 1990`. A value of '' '' overrides the value of *expire* previously set with `-e`.

`-h` *level* Specifies the security level at which the user can log in. Repeat the `-h` option as many times as necessary to specify every level at which the user may log in. This option is valid only if the Enhanced Security Extension is implemented.

`-v` *def_level* Specifies the default security level for the user. It must be one of the levels specified by the `-h` option. This option is valid only if the Enhanced Security Extension is implemented.

`-w` *hd_level* Specifies the security level for the user's home directory. This option is valid only if the Enhanced Security Extension is implemented.

`-a` *event* Specifies an event type or class (or a comma-separated list of event types or classes) that compose the user's audit mask. This option is valid only if the Auditing Extension is implemented.

login A string of printable characters that specifies the new login name of the user. It may not contain a colon (`:`) or a newline (`\n`) and must be unique.

RETURN VALUE

Upon successful completion, `useradd` returns a value of 0. Otherwise, it returns a non-zero value.

FILES

```
/etc/default/useradd
/etc/group
/etc/passwd
/etc/security/ia/ageduid
/etc/security/ia/audit
/etc/security/ia/index
/etc/security/ia/master
/etc/security/ia/level/logname
/etc/skel
```

SEE ALSO

defadm(BU_CMD), groupadd(AS_CMD), groupdel(AS_CMD), groupmod(AS_CMD), logins(AS_CMD), passwd(AS_CMD), passwd(AU_CMD), sh(BU_CMD), userdel(AS_CMD), usermod(AS_CMD), users(AU_CMD).

LEVEL

Level 1.

NAME

userdel – delete a user's login from the system

SYNOPSIS

`userdel [-r][-n` *months*`]` *login*

DESCRIPTION

The `userdel` command deletes a specified user's login definition from the system and makes the appropriate changes to login-related system files and the filesystem. The command also records the UID of the user being deleted in the `ageduid` file, so the UID will not be reused until a period of time has passed. The practice of keeping a UID out of use is called UID aging.

The following options are available:

`-r` Removes the user's home directory from the system. The files and directories under the home directory will no longer be accessible after this command is successfully executed.

`-n` *months* Specifies the number of months for which the UID should be aged. Specify `-1` to indicate the UID should never be reused. Specify `0` to indicate the UID may be reused immediately. If the `-n` option is not specified, the UID will be aged for a default number of months before it is reused.

login A string of printable characters that specifies an existing login on the system. It may not contain a colon (`:`) or a newline (`\n`).

RETURN VALUE

Upon successful completion, `userdel` returns a value of 0. Otherwise, it returns a non-zero value.

FILES

`/etc/default/userdel`	
`/etc/group`	
`/etc/passwd`	
`/etc/security/ia/ageduid`	
`/etc/security/ia/audit`	referenced only if the Enhanced Security Extension is implemented
`/etc/security/ia/index`	
`/etc/security/ia/master`	
`/etc/security/ia/level/`*logname*	referenced only if the Enhanced Security Extension is implemented

SEE ALSO

groupadd(AS_CMD), groupdel(AS_CMD), groupmod(AS_CMD), logins(AS_CMD), passwd(AU_CMD), passwd(AS_CMD), useradd(AS_CMD), usermod(AS_CMD), users(AU

NAME

usermod – modify a user's login information on the system

SYNOPSIS

`usermod` [`-u` *uid* [`-o`]] [`-g` *group*] [`-G` *group*[[`,`*group*] . . .]] [`-d` *dir*[`-m`]] [`-s` *shell*]
[`-c` *comment*] [`-l` *new_logname*] [`-f` *inactive*] [`-e` *expire*]
[`-h` [*operator*] *level* [`-h` *level* [. . .]]] [`-v` *def_level*]
[`-a` [*operator*] *event* [`,` . . .]] *login*

DESCRIPTION

The command `usermod` modifies a user's login definition on the system. It changes the definition of the specified login and makes the appropriate login-related system file and file system changes.

The options and arguments have the following meanings:

`-u` *uid* — Specifies a new UID for the user. It must be a unique non-negative decimal integer below `{MAXUID}`.

`-o` — This option allows the specified UID to be duplicated (non-unique). Because the security of the system depends on every UID being uniquely associated with a specific individual, never use this feature to assign more than one user the same UID.

`-g` *group* — Specifies an existing group's integer ID, or character string name. It redefines the user's primary group membership.

`-G` *group* — Specifies an existing group's integer ID, or character string name. It redefines the user's supplementary group membership. Duplicates are ignored. No more than `NGROUPS_MAX` groups should be specified.

`-d` *dir* — Specifies the new home directory of the user. This field is limited to `{PATH_MAX}` characters. It defaults to *HOMEDIR/login*, where *HOMEDIR* is the base directory for new login home directories and *login* is the new login.

`-s` *shell* — Specifies the full pathname of the program that will be used as the user's shell on login. This field is limited to `{PATH_MAX}` characters. The *shell* must be a valid executable file.

`-c` *comment* — Specifies any text string. It is generally a short description of the login, and is currently used as the field for the user's full name. It is limited to 128 printable characters and should not include colons (`:`).

`-l` *new_logname* — Specifies a string of printable characters that make up the new login name for the user. It may not contain a colon (`:`) or a newline (`\n`) and must be unique.

`-m` — Moves the user's home directory to the new directory specified with the `-d` option. If the directory already exists, the specified *login* must have access to it.

`-f` *inactive* The maximum number of days allowed between uses of a login ID before that login ID is declared invalid. Normal values are positive integers. A value of `-1` overrides the value of *inactive* previously set with `-f`.

`-e` *expire* The date on which a login can no longer be used; after this date, no user will be able to access this login. (This option is useful for creating temporary logins.) You may type the value of the argument *expire* (which is a date) in any format you like (except a Julian date). For example, you may enter `10/6/90` or `October 6, 1990`. A value of `" "` overrides the value of *expire* previously set with `-e`.

`-h` *operator level* Specifies the levels at which a user can log in. An operator can be any of the following three: `+` (to add), `-` (to delete), or `=` (to replace). The levels specified are processed based on the operator. If the operator is not given, the `=` (replace) operator is assumed. To add, delete or replace several levels, specify multiple `-h` options. However, an operator may be specified only on the first `-h` option; the same operation is assumed for any subsequent `-h` options. This option is valid only if the Enhanced Security Extension is implemented.

`-v` *def_level* Changes the user's default level to the one specified. This option is valid only if the Enhanced Security Extension is implemented.

`-a` *operator event*

Sets the user's audit mask based on the event(s) specified. An operator can be any of the following three: `+` (to add), `-` (to delete), or `=` (to replace). If an operator is not given, the `=` (replace) operator is assumed. This option is valid only if the Auditing Extension is implemented.

login Specifies a string of printable characters that make up the existing login to be modified.

RETURN VALUE

Upon successful completion, `usermod` returns a value of `0`. Otherwise, it returns a non-zero value.

FILES

`/etc/group`
`/etc/passwd`
`/etc/security/ia/audit`
`/etc/security/ia/index`
`/etc/security/ia/master`
`/etc/security/ia/level/`*logname*

SEE ALSO

groupadd(AS_CMD), groupdel(AS_CMD), groupmod(AS_CMD), logins(AS_CMD), passwd(AS_CMD), passwd(AU_CMD), useradd(AS_CMD), userdel(AS_CMD), users(AS_CMD).

LEVEL

Level 1.

NAME

volcopy, labelit – copy file systems with label checking

SYNOPSIS

`volcopy` [`-F` *FSType*] [`-V`] [*options*] [`-o` *specific_options*] *fsname special1 volname1 special2 volname2*

`labelit` [`-F` *FSType*] [`-V`] [`-o` *specific_options*] *special* [*fsname volume* [`-n`]]

DESCRIPTION

The command `volcopy` makes a literal copy of the file system using a block size matched to the device. The *options* and arguments for this command are:

`-F` *FSType* specify the *FSType* on which to operate. The *FSType* must be specified or must be determinable by searching an implementation-defined database for an entry matching the *special* specified.

`-V` echo complete command line. This includes additional information determined by a lookup in an implementation-defined database. This option is used to verify and validate a command line. The command is not executed.

`-o` specify *FSType*-specific options, if any

`-a` invoke a verification sequence requiring a positive operator response instead of the standard delay before the copy is made

`-s` (default) invoke the DEL if wrong verification sequence.

Other *options* are used only with 9-track magnetic tapes:

`-bpi`*density* bits per inch

`-feet`*size* size of reel in feet

`-reel`*num* beginning reel number for a restarted copy

`-buf` use double buffered I/O.

The program requests length and density information if this is not given on the command line or if it is not recorded on an input tape label. If the file system is too large to fit on one reel, `volcopy` prompts for additional reels. Labels of all reels are checked. Tapes may be mounted alternately on two or more drives. If `volcopy` is interrupted, it asks if the user wants to quit or wants to escape to the command interpreter. In the latter case, the user can perform other operations (e.g., `labelit`) and return to `volcopy` by exiting the command interpreter.

The *fsname* argument represents the file system name on the device (e.g., `root`, `u1`) being copied.

The *special* should be the physical disk section or tape and is hardware specific (e.g., `/dev/rdsk/1s5`, `/dev/rmt/c0s0`).

The *volname* is the physical volume name; it should match the external label sticker. Such label names are limited to six or fewer characters. The argument *volname* may be – to use the existing volume name.

The arguments *special1* and *volname1* are the device and volume, respectively, from which the copy of the file system is being extracted. The arguments *special2* and *volname2* are the target device and volume, respectively.

The command `labelit` can be used to provide initial labels for unmounted disk or tape file systems. With the optional arguments omitted, `labelit` prints current label values. The `–n` option provides for initial labeling of new tapes only (this destroys previous contents).

volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette, or tape.

USAGE

Administrator.

FUTURE DIRECTIONS

Support for the `-s` option will be eventually discontinued.

`volcopy` has been designated Level 2 as of July 1992. It will be removed from the SVID after the Level 2 period has expired. The functionality of `volcopy` has been replaced by `backup(AS_CMD)`.

LEVEL

Level 2, July 1992.

The following option has been moved to level 2 effective September 30, 1989:

`-s`

NAME

whodo – who is doing what

SYNOPSIS

`whodo [-h] [-l] [-z -Z] [`*user*`]`

DESCRIPTION

The command `whodo` produces merged, reformatted, and dated output from the `who` and `ps` commands [see `who`(AU_CMD) and `ps`(BU_CMD), respectively].

The `-h` option suppresses printing of the header.

The `-l` option displays a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The `-z` option prints the alias name of the level of each process. This option is only valid if the Enhanced Security Extension is implemented.

The `-Z` option prints the fully qualified level of each process. This option is only valid if the Enhanced Security Extension is implemented.

If *user* is specified, output is restricted to that user. If *user* is logged in on multiple terminals, all sessions for that user will be displayed.

A normal user is restricted to display only information on processes which the user owns and with a level that is dominated by the user's current level. An administrator is able to display information about all processes.

If the `-l` option is specified, the `-z` and `-Z` options are ignored. The `-z` and `-Z` options are mutually exclusive. If the `-z` option is specified and there is not an alias assigned to the level, the decimal value of the level identifier (LID) is displayed. If the `-z` or `-Z` option is specified and the level is in the *valid-inactive* state, the decimal value of the LID is displayed. LID states are described in `lvlname`(ES_CMD).

FILES

`/etc/passwd`
`/etc/ps_data`
`/proc/`*pid*
`/var/adm/utmp`

USAGE

General.

SEE ALSO

lvlname(ES_CMD), ps(BU_CMD), who(AU_CMD).

LEVEL

Level 1.

NAME

zdump – time zone dumper

SYNOPSIS

`zdump` [–v] [`–c` *cutoffyear*] [*zonename* `...`]

DESCRIPTION

The command `zdump` prints the current time in each *zonename* named on the command line.

The options and arguments have the following meanings:

`–v` For each *zonename* on the command line, print the current time, the time at the lowest possible time value, the time one day after the lowest possible time value, the times both one second before and exactly at each time at which the rules for computing local time change, the time at the highest possible time value, and the time at one day less than the highest possible time value. Each line ends with `isdst=1` if the given time is daylight saving time or `isdst=0` otherwise.

`–c` *cutoffyear*
Cut off the verbose output near the start of the year *cutoffyear*.

FILES

`/usr/share/lib/zoneinfo` standard zone information directory

USAGE

End-user.

SEE ALSO

ctime(BA_LIB).

LEVEL

Level 1.

NAME

zic – time zone compiler

SYNOPSIS

`zic` [`-v`] [`-d` *directory*] [`-l` *localtime*] [*filename* ...]

DESCRIPTION

The command `zic` reads text from the file(s) named on the command line and creates the time conversion information files specified in this input. If *filename* is `-`, the standard input is read.

The options and arguments have the following meanings:

`-v` Complain if a year that appears in a data file is outside the range of years representable by system time values (0:00:00 AM UTC, January 1, 1970, to 3:14:07 AM UTC, January 19, 2038).

`-d` *directory* Create time conversion information files in the directory *directory* rather than in the standard directory

`/usr/share/lib/zoneinfo`.

`-l` *timezone* Use the time zone *timezone* as local time. `zic` will act as if the file contained a link line of the form

link *timezone* localtime

Input lines are made up of fields. Fields are separated from one another by any number of white space characters. Leading and trailing white space on input lines is ignored. An unquoted sharp character (`#`) in the input introduces a comment which extends to the end of the line the sharp character appears on. White space characters and sharp characters may be enclosed in double quotes (`"`) if they're to be used as part of a field. Any line that is blank (after comment stripping) is ignored. Non-blank lines are expected to be of one of three types: rule lines, zone lines, and link lines.

A rule line has the form:

`rule` *name from to type in on at save letter/s*

For example:

```
rule  USA  1969  1973  -  Apr  lastSun  2:00  1:00  D
```

The fields that make up a rule line are:

name Gives the (arbitrary) name of the set of rules this rule is part of.

from Gives the first year in which the rule applies. The word `minimum` (or an abbreviation) means the minimum year with a representable time value. The word `maximum` (or an abbreviation) means the maximum year with a representable time value.

to Gives the final year in which the rule applies. In addition to `minimum` and `maximum` (as above), the word `only` (or an abbreviation) may be used to repeat the value of the *from* field.

type Gives the type of year in which the rule applies. If *type* is `-` then the rule applies in all years between *from* and *to* inclusive; if *type* is `uspres`, the rule applies in U.S. Presidential election years; if *type* is

`nonpres`, the rule applies in years other than U.S. Presidential election years.

in Names the month in which the rule takes effect. Month names may be abbreviated.

on Gives the day on which the rule takes effect. Recognized forms include:

`5`	the fifth of the month
`lastSun`	the last Sunday in the month
`lastMon`	the last Monday in the month
`Sun>=8`	first Sunday on or after the eighth
`Sun<=25`	last Sunday on or before the 25th

Names of days of the week may be abbreviated or spelled out in full. Note: there must be no spaces within the *on* field.

at Gives the time of day at which the rule takes effect. Recognized forms include:

`2`	time in hours
`2:00`	time in hours and minutes
`15:00`	24-hour format time (for times after noon)
`1:28:14`	time in hours, minutes, and seconds

Any of these forms may be followed by the letter `w` if the given time is local wall clock time or `s` if the given time is local standard time; in the absence of `w` or `s`, wall clock time is assumed.

save Gives the amount of time to be added to local standard time when the rule is in effect. This field has the same format as the *at* field (the `w` and `s` suffixes are not used).

letter/s Gives the variable part (for example, the S or D in EST or EDT) of time zone abbreviations to be used when this rule is in effect. If this field is –, the variable part is null.

A zone line has the form

`zone` *name* *gmtoff* *rules/save* *format* [*until*]

For example:

```
zone   Australia/South-west 9:30  Aus   CST  1987 Mar 15 2:00
```

The fields that make up a zone line are:

name The name of the time zone. This is the name used in creating the time conversion information file for the zone.

gmtoff The amount of time to add to UTC to get standard time in this zone. This field has the same format as the *at* and *save* fields of rule lines; begin the field with a minus sign if time must be subtracted from UTC.

rules/save The name of the rule(s) that apply in the time zone or, alternately, an amount of time to add to local standard time. If this field is –, then standard time always applies in the time zone.

format The format for time zone abbreviations in this time zone. The pair of characters `%s` is used to show where the variable part of the time zone abbreviation goes.

until The time at which the UTC offset or the rule(s) change for a location. It is specified as a year, a month, a day, and a time of day. If this is specified, the time zone information is generated from the given UTC offset and rule change until the time specified.

The next line must be a continuation line; this has the same form as a zone line except that the string `zone` and the name are omitted, as the continuation line will place information starting at the time specified as the *until* field in the previous line in the file used by the previous line. Continuation lines may contain an *until* field, just as zone lines do, indicating that the next line is a further continuation.

A link line has the form

```
link link-fromlink-to
```

For example:

```
link US/EasternEST5EDT
```

The *link-from* field should appear as the *name* field in some zone line; the *link-to* field is used as an alternate name for that zone.

Except for continuation lines, lines may appear in any order in the input.

Note: for areas with more than two types of local time, you may need to use local standard time in the *at* field of the earliest transition time's rule to ensure that the earliest transition time recorded in the compiled file is correct.

FILES

`/usr/share/lib/zoneinfo` standard directory used for created files

SEE ALSO

ctime(BA_LIB), time(SD_CMD).

LEVEL

Level 1.