

CHARLES RIVER DATA SYSTEMS

**UNOS Commands
Manual**

UNOS Commands Manual

CRDS: 97-21318-05

October 1986

Charles River Data Systems
983 Concord St.
Framingham, MA 01701

All Rights Reserved. No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying or recording on electronic media without written permission.

Copyright(c) 1986 Charles River Data Systems

UNOS is a registered trademark of Charles River Data Systems

UNIX is a registered trademark of Bell Laboratories

CRDS: 97-21318 Revision 05

Contents

- 1 **How To Get Started**
- 1 How To Use This Manual
- 2 Using UNOS
- 4 How To Use Commands
- 9 Basic Command Set

Commands

addname gives a file another name
advance increments eventcounts
ar archive and library manager for portable archives
archive maintains archive files
asm translates assembly language files
backup makes a backup copy of a file system
build selectively recompiles subsystems
calltree generates a call tree or cross reference
casm translates assembly language files to COFF object files
cat concatenates files
cc compiles C programs
ccom68 compiles C programs
cd changes working directory
cdeb debugs a program
change substitutes text strings in files
chgpw changes group password
chgrp changes the group associated with a file
chmod sets or changes modes on files
chown changes owner of files
chron performs periodic commands specified in file
cofftoel converts a COFF format object file to the old UNOS object file format
command executes a command line or file
common finds identical lines in two different files
compare compares two binary files
confedit creates and/or edits a UNOS configuration file
conftable displays either the contents of the current running configuration table or the contents from a configuration file

Contents

contig make a contiguous file
copy duplicates files
copy_boot copies a bootstrap program to a device
count counts items in files
cpw changes password
cut dissects files by fields or columns
date displays the system date and time
debe copies a file or a piece of a file
debug debugs a program
delete deletes files
describe displays manual page for a command or subroutine
difference indicates how two text files differ
diskusage counts blocks used by files
download loads and starts the IOP code
dump dumps selected parts of an object file
echo displays arguments
email sends and receives electronic mail
false returns an error indication
find locates files
fmtdisk formats a hard or floppy disk
format formats text files for printing
frexcv allows a remote Charles River machine to receive a file from a host
fscheck checks and fixes file and directory structures
ftrans allows a remote Charles River machine to send a file to a host
gcc compiles C programs using the Green Hills C compiler
genlib generates library entries from object programs and archives
getopt parses a command line
hexdec displays the contents of a file in various formats
hist constructs a histogram of function usage
id prints user and group IDs and names
installation reads files from a tape or disk in an installation file format
iostats executes a command and displays I/O statistics
ipstat displays status of IPC queues
kill sends a signal to processes
ksh, rsh shell, the standard/restricted command programming language
ld links editor for common object files
link binds object modules
list lists the contents of directories

log maintains logs of program or system changes
logger collects timing information
login verifies user ID and password, and starts top-level shell
lorder finds ordering relation for an object library
lpd runs the line printer from a queue
lpmaint maintains printer queue
lpq displays status of printer queue
lpr queues printing requests
makedev creates a device file system entry
makedir creates directories
makeec creates eventcount file system entries
makefs initializes a file system
match searches for matching text strings in files
message allows or prevents writes to terminal
mnetr receives a file via serial tty ports
mnett transmits a file via serial tty ports
mount mounts a file system
move renames files or moves them across file systems
namelist displays symbol table of object files
newgrp enters a new group
nice executes a command at a given priority
nm displays the symbol table for a COFF file
nohup runs a command immune to hangups and quits
notify runs program in background and reports completion
p paginates files and displays non-printing characters
package general purpose file dumping utility that writes files to a tape or disk in an installation (CPIO) file format.
paste combines files in columns
pmask displays the current file creation mode mask
performs special Command Interpreter functions
prfmt paginates files for printing
print_bad_blocks displays bad block list
print_partition displays the partition record for a hard disk
print_pvh displays a physical volume header from a disk
printec prints eventcount values
printenv prints out the values of environment variables
printlog displays system error messages
profile produces a function histogram
prtansi understands ANSI printer control characters
ps displays process status
pwd displays name of working directory
qfix repairs corrupted queues

Contents

qmaint controls processing of a queue
qprint lists the contents of a queue
quicksort sorts the contents of files
reboot reboots the operating system
reltocoff converts a file from the old UNOS object format file to COFF format
repeat executes a command repeatedly
restore recreates a file system from a backup
resume resumes execution of a suspended process
setdate sets system date and time
setmask sets the file creation mode mask
setpri sets priority of a process
setuser changes the current user and group ID
shutdown brings the system to single user mode
size prints section sizes of common object files
sleep delays execution for a specified interval
space displays amount of free space in the file system
strip strips symbol and line number information from a common object file
su changes the current user id
suname sets system, node, and machine names
suspend halts execution of a process
sync updates disk with in-memory information
sysgen is a program which links system file with the specified input object files and updates table *devsw* of the linked file.
sysversion displays UNOS version date
tail displays the first or last lines of a file
ted creates and edits files
tee duplicates output in files
term asynchronous terminal emulator
time displays execution time for a command
tp performs magnetic tape operations
translit substitutes characters
true returns a success indication
ttymodes sets or displays terminal modes
uget retrieves files from a Version 6 UNIX-style file system
umount unmounts a file system
uname print name of current UNIX system
unboot is the UNOS bootstrap.
unique displays repeated or unique lines in files
umount unmounts a file system
update periodically writes out disk cache

uptime shows how long system has been running
v7uget retrieves files from a Version 7 UNIX-style file system
version displays creation date and time of object files
wait waits for process to exit
wall sends a message to all users currently logged on
wcc compiles C programs using the Whitesmiths C compiler
when executes a command line at a given time
whenq inspects queue of **when** requests
whenrun runs queued **when** requests
which shows which file will be executed for a name
who displays names of users currently logged on
write sends a message to specified users
write_bad_blocks creates/overwrites bad block list on disk
write_partition creates/overwrites the partition record for a hard disk
write_pvh places a physical volume header on the specified disk device.
yes writes a word repeatedly to standard output

Commands Functional Listing

How To Get Started

This introduction provides a general overview of what you need to know to get started on UNOS. Specifically, we discuss

- Using UNOS
- How to use commands
- Basic command set

For a more detailed introduction to UNOS, see the *User's Guide to UNOS*. A *Commands Functional Listing* appears at the end of this manual.

How To Use This Manual

Throughout our manuals we employ the following typesetting conventions:

<cr> indicates a carriage return (the RETURN key)

^X indicates a control character (the CTRL key plus a character) typed in a single keystroke

\$X indicates an escape sequence (the ESC key followed by a letter) typed sequentially

\$ indicates the system prompt

indicates superuser prompt

indicates a number: 5, 10, 128 etc.

[] indicates that anything enclosed (within the brackets) is optional

All system messages appear in **typewriter bold** and command names appear in **bold type**, while user input appears in typewriter type. All commands end with a carriage return, even though most examples do not indicate the carriage return.

How To Get Started

Using UNOS

In the following sections, we discuss:

- How to Log On and Log Out
- How to Communicate Through Your Terminal

How to Log On and Log Out

To log on, obtain a valid user name and, optionally, a password from your system administrator.

Turn on the terminal and press the RETURN key until you see the **Name:** prompt. The **Name:** prompt indicates the system is up and the terminal is ready to accept input. Type your user name; you must type it exactly as your system administrator gave it to you. Then press the RETURN key. You'll see the **Password:** prompt.

If you have a password, type it and press the RETURN key. If you don't have a password, simply press the RETURN key. Note that your password does not appear on the screen.

The **Name:** prompt reappears if you type a user name and password that differs from the user name and password your system administrator gave you. The **Name:** prompt keeps appearing until you type in the correct user name and password.

When you successfully log on, the message of the day and notification of any new mail appear. Following the messages, the system prompt **\$** appears indicating that the system is ready to accept commands.

To log out, type a **^D** after the system prompt **\$**. See the *Control Characters* section for more information on **^D**. You'll see **<EOF>** echoed. Then the login prompt **Name:** will appear on your screen.

How to Communicate Through Your Terminal

Set your terminal at full duplex (see the *UNOS System Concepts Manual* for details). The system saves any characters you type (except control characters) until you press the RETURN key. Pressing the RETURN key delivers the characters to the Korn Shell, the default shell on UNOS. You can type ahead; the system prompt \$ doesn't have to appear before you begin typing the next command.

The DELETE key corrects typing errors. DELETE erases the character to the left of the cursor.

Control Characters

Control characters perform special functions at the terminal. To use a control character, press the CTRL key and another key simultaneously. A description of the control characters and their functions follows:

- ^C** kills the command being run, and depending on the command, returns the system prompt \$.
- ^D** clears the line, when typed in the middle of a command line, otherwise it ends a file. As an end of file indicator, **^D** signals the end of standard input or logs you out.
- ^Q** cancels the effect of **^S**, continuing the display.
- ^R** redisplay the current unfinished input line. This can be useful when input and output mesh on the screen.
- ^S** suspends output to the terminal, freezing the display information. If a program generates rapid output, **^S** can stop output so you can examine a screenful of information at a time.
- ^U** erases the current line and displays no new prompt, \$
- ^Y** suspends execution of a command. You use **^Y** to debug programs.

How To Get Started

How to Use Commands

When you have successfully logged on to UNOS, the program monitors your terminal. The shells reads typed-in lines, splits them into a command name and arguments, and executes the command. A command is simply an executable program. To find a command, the Command Interpreter looks first in your current directory. If there is no match, it looks in **/bin**, and then in **/usr/bin**.

When a program terminates, the shell regains control and returns the system prompt **\$** to indicate that it is ready for another command. The *User's Guide to UNOS* describes other capabilities of the Command Interpreter.

Command Format

This section discusses the types of command formats. These types describe general statements about command arguments.

The command name is the first word on an input line to the shell. Separate the command and its arguments with white space, a space or tab characters.

Commands take the following formats:

1. **a command with no arguments**
\$command

For instance, to display the current day and time, type
\$date

2. **a command with a dash argument**
\$command -argument

For instance, to display the long listing of the directory, type
\$list -long

You can abbreviate arguments if the particular command allows abbreviation. For example, the command **list** allows the argument **-l** to represent the the argument long.

```
$list -l
```

3. **a command with an equal argument**
`$command argument=argument`

For instance, to send the file **newsletter** as a mail message, type

```
$email send=newsletter
```

4. **a command with one or more arguments**
`$command argument1...argumentn`

If a command takes several arguments, separate them with a space. For instance, to list the long format of all files in a directory, type

```
@>list -a -l
```

To list the long format of all files that have the word *chapter* in their name and display the files in order of last modification, type

```
$list -l -time -r *chapter*
```

or

```
$list -l *chapter* -time -r
```

If a command takes two file type arguments, the first is often the input and the second, the output. For instance, to copy the contents of *filename* to *chap.01*, type

```
$copy filename chap.01
```

With some commands, such as **prfmt**, **lpmaint**, and **format**, the argument applies only to the filename that follows immediately. For instance, to print *file1* with a brief footer and *file2* with a page length of 40 characters, type

```
$prfmt -b file1 l=40 file2
```

How To Get Started

In most cases, arguments can appear in any order, anywhere on the command line, following the command name. The next section discusses combining arguments. However, there are a few exceptions:

- Some commands (such as **nice**, **notify**, **repeat**, **time**, and **command**) expect another command as argument. For instance:
`$time list`
- If you use a dash or an equal argument with these commands, the argument must follow the command to which it refers. For instance:
`$repeat delay=60 ps -a`
The *delay=60* argument refers to the **repeat** command and the *-a* argument refers to the **ps** command.
- If you want to use an equal sign in a command line without having it interpreted as a flag argument, precede the equal sign with a backslash or two dashes. For instance
`$change filename\=temp filename_temp myfile`
or
`$change -- filename=temp filename_temp myfile`

Input/Output Redirection

Most commands read from standard input (the keyboard) and, write to standard output and standard error (the screen). To read from nonstandard input and to write to nonstandard output and nonstandard error, you must redirect your input and output. You can redirect input or output in these ways:

1. < standard input

When you use **<**, the standard input comes from the designated file. `$change "test" "text" < tmp`

This command line uses the file *tmp* as input, displays the output -- the result of the change -- on the terminal.

2. > standard output

When you use **>**, the system sends the standard output to a designated file, creating a new file if necessary.

```
$list *.doc > fileout
```

This command line sends a listing of files with the suffix *.doc* in the directory to the file **fileout** instead of displaying it on the terminal.

Append output to a designated file by using **>>**.

3. **% standard error**

When you use **%**, the system sends the standard error output to the designated file.

```
$sys/lpd centronics %/tmp/c.error&
```

This command line starts the centronics daemon and directs its error messages to the file **/etc/c.error**

Append error output to a designated file with **%%**.

4. **| pipes**

Pipes connect the output of one program to the input of another.

```
$list | count -l
```

This command line executes two commands, **list** and **count**. The list of files provides input for the count program.

5. **& background**

The ampersand (**&**) disconnects the program from your terminal runs it asynchronously in the background.

```
$match "fileopen" *.doc > fileout &
212
$
```

This command line matches all instances of the word *fileopen* in the files with the *.doc* suffix, and places the results in the file *fileout*. The ampersand indicates a command running in the background. The process id and the system prompt appear on the screen after you type the command. You do not need to wait for the command to complete before executing other commands. Use the **ps** command to find out when the command line finishes executing.

Regular Expressions — Pattern Matching

The shell uses certain characters with special meaning for pattern matching.

Under UNOS, there are two types of pattern matching:

1. The shell expands the special pattern matching characters.

The shell replaces the specified pattern with any number of matching file names from the current or specified directory. For instance, if you execute the following command lines, the shell will expand the special character(s) with matching file names from the current directory:

```
$list *  
$echo memo.?  
$prfmt doc?.*
```

2. Commands (programs) use the special characters as an argument.
Since arguments pass to the commands, you must quote the special character or the character may expand before it passes. Only a few commands take pattern matching characters as arguments (i.e., **match**, **change**, **translit**).

Find a list of the special pattern matching characters in the *UNOS Command Manual*. Note that with some of the characters, the action depends on whether the shell or a command uses the character.

Also note that the shell does not match filenames that begin with a dot (.) unless the pattern explicitly begins with a dot (i.e., use .* not just *).

Numbers and Ranges

Commands, such as **cut**, **fscheck**, and **unique**, take ranges. A range is a list of numbers separated by commas. For example, 2,4,6,8 means items 2 and 4 and 6 and 8. You can use a dash to

indicate number ranges. To request a range from the first number indicated to any number place a dash before the number. You can request a range from a number to the end number by placing the dash after the number. For the range in between a numbers, place the dash between the numbers. For example:

```
$unique f=1-3,5,7-9 test
```

Checks for field uniqueness on fields 1,2,3,5,7,8 and 9.

```
$cut f=5- test
```

Cuts from field 5 to the end.

On-line Documentation

UNOS has two types of on-line documentation. Each command has a **-help** option that displays a command's usage line.

```
#list -help
```

Usage: list [options] [file1...filen]

Options: -a -dir -fileno -group -l -m -nosort -p -r -time

If you want more detailed information, use the **describe** command followed by the name of the command. For example, this command line displays the manual page for the **list** command.

```
$describe list
```

Basic Command Set

The following is a list of the minimum commands necessary to begin using the system. The commands appear alphabetically in this manual and should be consulted for more information on what they do and how to use them.

cd changes working directory

change substitutes text strings in files

common finds identical lines in two different files

copy duplicates files

count counts items in files

delete deletes files

describe displays manual page for a command or subroutine

How To Get Started

difference indicates how two text files differ
echo displays arguments
email sends and receives electronic mail
lpr queues printing requests
list lists the contents of directories
makedir creates directories
match searches for matching text strings in files
move renames files or moves them across file systems
p paginates files and displays non-printing characters
ps displays process status
pwd displays name of working directory
ted creates and edits files
translit substitutes characters
who displays names of users currently logged on
write sends a message to specified users

addname

addname gives a file another name

FORMAT

addname newname oldname

DESCRIPTION

addname creates an additional name, or alias, by which a file can be referenced. One file can have several aliases.

oldname must be a complete pathname, relative to the position of *newname*. You can then use *newname* to reference the file, without always using the complete pathname.

There is no way to distinguish an alias from a file's original name.

To disassociate an alias from a file, use the **delete** command to delete the alias. The file will still exist; the name will not.

When you delete one alias for a file that has many aliases and create a new file with that same name, the remaining aliases still refer to the original file; they are not associated with the new file. However, when you alter the contents of a file with multiple aliases, all the aliases are still associated with the same file and reflect its new contents.

OPTIONS

none

EXAMPLES

To give another name, *name2*, to the file *foo*, type
`$addname name2 foo`

DIAGNOSTICS

Cannot **addname** across file systems.

Can't **addname** 'newname' to 'oldname'.

addname

You cannot use **addname** across file systems.

ALSO SEE

copy duplicates files

delete deletes files

list lists the contents of directories

move renames files or moves them across file systems

NOTES

You cannot use **addname** across file systems, but you can have aliases in different directories within the same file system.

Most programs that take more than one filename as an argument do not recognize aliases. In addition, **archive**, **change**, **genlib**, **ted**, and **copy -r** can change the status of an alias.

Any single file can currently have a maximum of 127 names. Use the **list** command with the **-l** option to see how many aliases a file has; the number of aliases is indicated by the number in parentheses, in the column before the owner of the file.

USES

nothing

advance

advance increments eventcounts

FORMAT

advance [*options*] event_count_name1...event_count_namen

DESCRIPTION

advance increments each specified eventcount's first counter by one. If the *-verbose* option is used, **advance** prints the new value on the terminal. The print format is:

Eventcount "foo": counter one is 10, counter is 15.

When the command is done, you'll see an exit message that varies depending on what was accomplished:

- If no eventcounts are advanced you'll see the message:
No eventcounts advanced.
- If one eventcount was advanced, you'll see the message:
One eventcount advanced.
- If two or more eventcounts were advanced, you'll see the message:
2 eventcounts advanced
(or 3..4..5...etc)
- If one eventcount was not advanced, you'll see the message:
One eventcount not advanced.
- If two or more eventcounts were not advanced, you'll see the message:
2 eventcounts not advanced.
- If some eventcounts were advanced and others were not, you'll get two messages telling how many were advanced and how many were not.

Use the *-silent* flag to avoid printing exit messages.

advance

OPTIONS

You can abbreviate all options to their first letter.

-interactive

Prompts the user for more eventcounts after advancing any eventcounts given on the command line. Press the RETURN key (or ^D) on a blank line to end the prompting.

-help

Prints, to standard error, a few lines listing all the command options **advance** accepts.

-query

For each eventcount specified on the command line, asks the user to verify that the eventcount should be advanced.

-silent

Prints no messages to standard out except in case of error. No exit message is printed.

-verbose

Lists eventcounts being advanced, in the form explained under the **DESCRIPTION** section.

EXAMPLES

To shut down the system, type
`$advance /etc/shutdown.ec`

NOTE: It is safer to use the **shutdown** command.
Normally, ADMIN only should have write access to this eventcount.

DIAGNOSTICS

advance: An eventcount-type operation was specified on a non-eventcount logical unit.

"Bad flag: '-badflag!'"

Error encountered while reading from stdin.

Cannot open 'eventcount name'.

Cannot advance 'eventcount name'.

Cannot read eventcount 'eventcount name'.

advance

Cannot read second counter of eventcount 'eventcount name'.

ALSO SEE

printec prints eventcount values

NOTES

If no eventcounts are specified, **advance** expects a list of filenames, one per line, to be provided on standard input. Unless the *-interactive* option is used, no prompts are given. This means, for instance, that the following command is valid:

```
$list * | advance
```

USES

nothing

ar archive and library manager for portable archives

FORMAT

ar key [*position_file*] archivefile [*names*]

DESCRIPTION

ar maintains groups of files by combining the individual files into one larger file, or archive. To manipulate the contents of these archive files, **ar** maintains headers within them that have been standardized for all versions of **ar**. This allows the headers to be ported to different machine archetects.

The most common use of **ar** is for the maintenance of libraries used by the link editor. The link editor uses information stored in archive files so that it can efficiently manage multiple passes over libraries of object files. For more information concerning archival formats consult the *UNIX Programmer Reference Manual* **ar(4)**.

Commands to **ar** comprise keys, filenames used as references for positioning, archive files, and names of files on where actions take place. *archivefile* refers to the archive file to be manipulated. *position_file* refers to the filenames used as positioning references and *names* specify files upon which actions should take place. *keys* and *options* are described further below.

OPTIONS

Precede *keys* with an optional dash (-). Valid keys are

- d* Delete named files from the archive.

- m* Move named files to the end of the archive. If key *m* is used in conjunction with one of the positioning options, *a*, *b*, or *i*, then it must be accompanied by a *postion_file* to indicate that new files should be placed before (for *b*) or after (for *b* and *i*) *position_file*.

- p* Print the named archive files.

ar

- q* Quickly append the named files to the end of the archive. *Q* does not allow the use of positioning keys nor does it check to see whether added members are already present in the archive.
- r* Replace the named files in the archive file. If key *r* is used in conjunction with the option *u*, then move only files whose dates of modification are later than the archive file. Positioning keys can be used in conjunction with this key in a similar manner to the way they are used with key *m*.
- t* Print a table of contents for the archive file. If no names are specified, the entire archive file is catalogued. If names are included, the table will comprise only those files.
- x* Extract the named files. If no names are specified, extract all files. The *x* key does not alter the contents of the archive file.

The following keys can be combined with the keys described above to alter the manner in which they are carried out.

- c* Suppress the message **ar** prints when an archive is created.
- l* Places temporary files in the current working directory instead of the directory stipulated by the environmental variable TMPDIR or in the default directory **/tmp**.
- s* Forces the regeneration of the archive symbol table even if **ar** is not invoked with a command that modifies the archive files themselves. This key can restore symbol tables that were removed with the **strip** command.

EXAMPLES

The following command prints a table of contents for an archive:
shell> ar t myarchive

DIAGNOSTICS

Self-explanatory

ALSO SEE

ld links editor for common object files

lorder finds ordering relation for an object library

strip strips symbol and line number information from a common object file

NOTES

If an argument names the same file on more than one occasion, it may be archived more than once.

USES

/tmp/ar* temporary files

archive maintains archive files

FORMAT

`archive [options] archivefile...n [file1...n]`

DESCRIPTION

archive creates and maintains archive files. Archive files contain a group of individual files and a directory to those files. You can use them any where it is desirable to have a group of files treated as a single unit. For example, you can use them to simplify backup procedures, or to create and maintain object libraries for use by the linker.

OPTIONS

You can use only one of the following options at a time: *-d -l -p -u -x*.

You can use the *-setowner* and *-v* options in conjunction with any other option.

-d Deletes file(s) from the archive.

The archiver deletes all files in the input list. You must specify at least one file.

-l Lists the contents of the archive on standard output.

If you specify any file names, only those files are listed. If you use the *-v* option, the archiver list the size, modes, owner, and modification listed in addition to the file names.

-p Prints files.

The archiver writes a copy of each named file to standard output. You can use this option to examine files in the archive, or to extract a file under a different name by redirecting standard output. If you do not specify any files, the archiver prints all the files in the archive.

-setowner Restores the owner and group.

When you use this option, the archiver sets the owner and

archive

group of the extracted file to the original owner and group of the file. Only the system administrator can use this switch.

-u Update the archive.

The archiver inserts all files in the input list into the archive. If a file of the same name already exists in the archive, it is replaced by the new file. If the archive file does not exist, it is created.

-v Verbose.

If you use this option with the delete, update, or extract options, the archiver displays each operation on each file. When you use this option with *-p*, the archiver prints each file name. When you use it with *-l*, the archiver prints all information about the archive and each file.

-x Extracts file(s) from the archive.

The archiver extracts each file in the input list and puts it into a separate file with its original name and access modes. If you do not specify any files, the archiver extracts all the files in the archive.

Note that the archive file is not altered by this operation.

EXAMPLES

To delete the file *foo.c* from the archive file *archfile.a*, type
`$archive -d archfile.a foo.c`

To list the contents of the archive file *archfile.a*, type
`$archive -l archfile.a`

To extract the file *foo.c* and put it in the file *foo.x*, type
`$archive -p archfile.a foo.c > foo.x`

To update the archive file *archfile* with files *foo.c* and *bar.c*, type
`$archive -u archfile.a foo.c bar.c`

To extract the file *bar.c* from the archive file *archfile.a*, type
`$archive -x archfile.a bar.c`

DIAGNOSTICS

The archiver prints an error message if it cannot open a file in the input list (on updates) or if the file is not in the archive (all other operations). After it prints the error message, it continues processing the other files.

All other errors cause the archiver to terminate with an error message, leaving the archive file unchanged.

ALSO SEE

nothing

NOTES

On an update operation, while you can specify pathnames for input files, the archiver only stores the file name. On operations that refer to the archive file, pathnames have no meaning. When you extract files from the archive file, the archiver places the files in your current directory. Use the *-p* option if you want to either rename the file, or place it in another directory.

USES

/tmp/archfile.pid

Holds files during update and delete operations.

/tmp/archdir.pid

Holds the old directory

.archtemp.pid

In the same directory as the archive file, holds the new directory.

asm translates assembly language files

FORMAT

asm [-g][o=file][-s] file1...filen

DESCRIPTION

asm assembles the concatenation of the named files.

OPTIONS

-g Global.

Treats all undefined symbols in the assembly as global (used by the C compiler).

o=file Output file.

Leaves the output of the assembly in the specified file, instead of in the default *t.out*.

-s Short.

Forces all absolute addresses to sixteen bits.

EXAMPLES

To put the output of the file *mch.m* into the file *mch.j*, type

```
$asm o=mch.j mch.m
```

DIAGNOSTICS

The following is a list of the possible diagnostics:

Can not open <filename>

The assembler cannot open the source file.

Extra characters on line

Non-blank characters are found after the end of an assembler instruction.

Word at odd location

An attempt was made to assemble a word value at an odd

asm

location. This is probably due to a missing 'even' after a **.byte** or **.ascii** command.

<symbol> multiply defined

The <symbol> is defined twice with different values. This can also occur if the value of the <symbol> changes between the first and second passes of the assembler (phase error).

Bad assignment to '.'

An assignment is attempted, which would set '.' to a value smaller than the current location, or the value to be assigned is in a different segment (.text or .data) than the current one.

Missing op

An operator is missing in an expression.

<symbol> undefined

The <symbol> named is not defined in this file or is not marked as a global.

Operand type clash

An arithmetic operation is specified on operands of incompatible types.

Division by 0

Division by 0 is attempted in an expression.

Missing <char>

The specified character is expected but is not present. This is usually due to unbalanced parentheses.

Register required

A register name is required in the address mode.

Illegal type

An illegal address mode is specified for an instruction. Check the *68000 User's Manual* for legal addressing modes.

Wrong number of operands

The wrong number of operands is given for a 68000 instruction. The *68000 User's Manual* specifies the correct number of operands.

Too many operands

The number of operands given on the source line exceeds the maximum allowed by the assembler.

Bit number out of range

The bit number in a static bit instruction is outside the range 0-7.

Illegal addressing mode

An address register cannot be used as a byte-size operand.

Illegal size for memory shift

Memory shift instructions must be word-size.

Illegal trap value

Trap values must be in the range 0-15.

Symbol required

A symbol name is required, as in the .global operator.

Illegal string delimiter

An invalid delimiter is specified in an .ascii or .asciz command.

.ascii past end of line

The string in the .ascii or .asciz command continues past the end of the line. Check for a missing delimiter.

Absolute value required

A non-relocatable value is required.

Byte overflow

A value is specified that is too big for a byte.

Word overflow

A value is specified that is too big for a word.

Quick or count too large

The immediate value is too big for an addg, subq or shift instruction.

asm

ALSO SEE

link binds object modules

UNOS Assembler Notes

NOTES

The assembled file is not normally executable. It is executable only if there were no unresolved external references and there was no *.data* section in the program.

The linker relocates references to the data section to a known address (0x400000) which the system sets up at run-time via the memory management system. (This address is hardware dependent.)

USES

nothing

backup

backup makes a backup copy of a file system

FORMAT

backup [*options*] *dev* *backup_dev*

DESCRIPTION

backup reads the file system on the specified device (*dev*) and writes the files, in a special format, to the specified output device (*backup_dev*). The default is to backup all the files on the file system. You can specify that selected files are backed up according to their revision dates (see options).

backup counts the number of output volumes required, announces the number, and asks if you want to proceed. Then, as each volume is needed, **backup** requests mounting the volume and waits for it.

OPTIONS

-after date

Backs up files whose revision date is after the specified date. The format for date is hhmm, MMDDhhmm, or MMDDhhmmyy where MM is the month number, DD is the day of the month, hh is the hour, mm is the minute, and yy is the year.

NOTE: This is the same format as the **setdate** command.

The default value is the beginning of time, midnight Jan. 1, 1980.

-before date

Backs up files whose revision date is before the specified date. The format for date is the same as for *-after*. The default value is the time **backup** starts.

-bf#

Blocking factor for output records, in disk blocks.

The output record size is determined by blocking factor * disk block size. If *backup_dev* is a disk, omit *bf* so that speed optimizations can be made.

backup

-id string

Stores the string in the backup as an identifier.

Use this option, for example to record which file system the backup is for and whether it is a full or partial backup. The string can be up to 80 characters long, but cannot contain blanks, tabs, or newlines.

-nospan

Does not allow files to span volumes.

-rs#

Restarts a backup from volume <number>.

Use this option only if the file system you are backing up has not changed! Volume <number-1> will be requested and some information read from it. The backup will then proceed from volume <number>. Enter all other options as they were originally specified.

-vs#

Volume size of *backup_dev* in output records.

If *backup_dev* is a disk, set the volume size to the number of blocks on the disk.

EXAMPLES

To backup all files on device **/dev/rcc0** to floppy disks on **/dev/rcfd0** (*vs* is set for double-sided/double density floppys):
\$backup vs=1989 id=cc0_full /dev/rcc0 /dev/rcfd0

To backup files on device **/dev/rcc0** with revision dates after midnight April 14th, type
\$backup vs=1989 after=04140000 id=cc0_incremental /dev/rcc0 /dev/rcfd0

DIAGNOSTICS

The messages are all self-explanatory.

ALSO SEE

restore recreates a file system from a backup

backup

NOTES

The output file must already exist and it must be a device node.

Use the raw input and output devices for optimum speed. Make certain that there is no other activity on the file system while doing back-ups. Otherwise a file could be backed-up in an inconsistent state.

USES

nothing

build

build selectively recompiles subsystems

FORMAT

build [*mf*=filename][*-s*] *object1*...*objectn*

DESCRIPTION

build lets you execute command sequences based on the modification dates of files. **build** does this by recognizing the dependencies of objects on each other. **build** objects are pathnames either of files or macros. **build**'s objects are often, but not always, the same object files created by the compiler. This documentation uses the term *object* to mean a **build** object, unless it refers specifically to object files.

build always assumes you are building the specified objects in your current working directory.

To execute **build** you must also use a *buildfile* to specify the activity that the command will execute. **build** determines its object dependencies from an internal table it creates with specifications in the buildfile. The internal table allows **build**, for example, to create a *.o* file from a corresponding *.c* or *.s* file.

OPTIONS

mf=*filename* Specifies the buildfile.

You may specify up to 32 buildfiles. The default buildfiles are the file *Buildfile* in the current directory. **build** reads buildfiles in the order of their specification in the command.

-s Suppresses normal operation messages that would appear on the screen.

object The specific object within the buildfile that **build** should create. The object specified on the command line must be a filename, not a macro.

build

build creates an object if the object is specified on the command line, or if **build** must use the object to build another object. Objects are built in the order specified in the command or in its buildfile's internal dependency lists (described below).

build considers an object as "building" from the time **build** begins creating that object. No object can finish building until all of the objects upon which it depends are built. It is an error to attempt to build an object already being built by the same **build** command. (**build** is depth-first on the partial order.) No object is built more than once.

Buildfile Description

The format of a Buildfile is a series of transition descriptions. Each transition description consists of an object (called an *item* in this position) followed by an optional list of objects on which the item depends (thus, a *dependency list*). The order of transition descriptions within a buildfile is unimportant, though we recommend the order we describe later.

These general terms are essential to understanding transition descriptions:

- item list
- dependency object list
- action line
- macros

An *item list* may have up to 100 comma-separated objects. **build** ignores ends-of-line following any comma in the list. Each object in the item list is associated with an action line (see below) and with the dependency object list of that transition description. No object may appear as an item in more than one transition description within the same buildfile. If this happens, the last definition of the object overrides any dependency object and action lists associated with that object in an earlier-specified item list.

The *dependency object list* is a possibly empty list of objects upon which the objects in the item list depend. Macros may be used in the list. The entire list must be on one logical line. List members must be separated by spaces or horizontal tabs.

build

Action line(s) consists of command lines with initial tabs. The next line that does not begin with a tab is not part of that line. The initial tab of each action line is ignored, but subsequent tabs in the line convert to spaces. If no action line is specified in a transition description **build** will apply the suffix rules (described later).

macros in buildfiles are internally-defined names that group objects logically for **build** operations. A macro may be used in an item list or a dependency object list.

The following sample buildfile would be used by the command:
\$ build program

```
$objdir=../obj
$searchlist=/src/sys/drivers ../obj\
            /srcs/header no_obj /include
.o = .c .s
        cc -c $0 casm o=$*.o $0
""= .o

$errlog.o: errlog_formats.h errlog_sources.h errlog.h

program:  member1.o member2.o member3.o $errlog.o
        cc -w -c -DCP32 -DPHYS

list1,
listing:  member1.c member2.c member3.c
        pr member1.c member2.c member3.c | lpr
        date >listing
```

NOTATION

Buildfiles use the following special symbols:

\ (backslash)

continues the end of a buildfile line onto the next line, ignoring the end-of-line (EOL) marker. (Important for reading buildfiles easily.)

, (comma)

separates multiple items defined by the same dependency list.

build ignores end-of-line characters after any comma in an item list.

build

(space)

whitespace, spaces or horizontal tabs act as object separators in the dependency object list. Space is ignored within the item lists.

= (equals) or

: (colon)

sets off an item name(s) from the dependency list.

&& (double ampersand)

a special “and” notation in action lines.

|| (double pipe)

a special “or” notation in action lines.

\$ (dollar sign)

is a special macro prefix.

\$<item> or

\$<component>

notifies **build** that the item or component is a macro.

\$() (zero)

a special macro for the object directory pathname (under suffix rules).

\$@ (at sign)

a special macro for the object being built.

\$* (asterisk)

a special macro for the object being built, stripped of its suffix.

\$<number>

in an action line, substitutes a file in <number> position of the dependency list

\$r<number>

in an action line, substitutes a file in <number> position **and all subsequent files** of the dependency list

STRUCTURE

There are five different types of buildfile transition descriptions:

- **\$objdir** (main object directory)
- **\$searchlist** (directories to search for objects)
- Suffix rules
- Macro definitions
- Modules
 - * Module item (name) and dependency object lists
 - * Action line (optional)

Transition descriptions consists of an item (or items), followed by a colon or an equals sign, followed by a dependency object list, followed by an end-of-line. Each type of transition description has its own requirements, as described in the following sections.

\$objdir and \$searchlist

These two macros define special transition descriptions. The example uses the following **\$objdir** and **\$searchlist** definitions:

```
$objdir=../obj
$searchlist=/src/sys/drivers ../obj\
             /srcs/header no_obj /include
```

\$objdir specifies the object directory for the source files found in the current working directory. The **\$objdir** name is a required keyword. When **build** executes an action line to create an object it expects to find the object in the object directory. The first and only object in the dependency object list of **\$objdir** is that directory. The default directory is the working directory. The object directory is always part of the **\$searchlist**.

\$searchlist tells the **build** command where to look for objects in the directory tree if the files have unknown directories.

\$searchlist's dependency object list consists of pairs of alternating object and source directories, with the object first. The working directory and the **\$objdir** are always the first pair, preceding any of the specified pairs. **build** only searches for object directories for objects with a specified action list. The searches go no further in the searchlist than the closest pair sought after for the dependency object. An object found in the object directory builds by the execution of action lines.

build

NOTE: Do not use **\$searchlist** to search for objects with a "/" in their name. A "/" indicates objects that are either absolute or relative to the working directory.

Suffix Rules

build always wants to find an action line for each object. It uses *suffix rules* as default action lines for all objects in a buildfile that have no specific action line. Because **build** is most often used in program development, it is sensitive to file suffixes, such as `.c` for source files, `.s` for assembler files, and `.o` for object files. The understood relationships between files with are defined in the default suffix rules:

```
.obj = .pas .for
      pascal -obj $0
      fortran -obj $0

.j = .c .m
    cc -c $0
    asm o=$*.j $0

"" = .j .obj
    cc o=$* $r0
    fortran o=$* $r0

.o = .c .m
    cc -c $0
    asm o=$*.o $0
```

The example uses the suffix rules:

```
.o = .c .s
    cc -c $0 casm o=$*.o $0

"" = .o
```

You may also specify your own suffixes for **build** flexibility. See the *Build Process* chapter in the *UNOS Developer's Manual* for more information.

build

Suffix rules use macros with numbers for greater flexibility. When **build** constructs an item and finds an object with a specified suffix rule, that object becomes the “zeroeth” object of that item. The n'th suffix corresponds to the n'th command. An item can have a null suffix (“”), but any object to be built with a null suffix rule must not be associated with a dependency object list, or no suffix rule will be applied.

Macro Definitions

You may define any item as a macro for use in subsequent dependency object lists. Macros use the standard transition description format, but generally do not have action lines. They generally use the suffix rules for execution information. The following line in the example defines a macro:

```
$errlog.o: errlog_formats.h errlog_sources.h errlog.h
```

Modules

Buildfile modules perform the actual operation requested by the **build** command, usually through action lines. There are two modules in the example buildfile.

```
program: member1.o member2.o member3.o $errlog.o
        cc -w -c -DCP32 -DPHYS
list1,
listing: member1.c member2.c member3.c
        pr member1.c member2.c member3.c | lpr
        date >listing
```

The module item should be an actual file so it can be specified with the **build** command.

Action line commands execute after **build** creates the objects in the dependency object list. To build the module **program** in the example, **build** performs the following steps:

1. Checks the modification times of member1.c member2.c and member3.c.

build

2. If any of the times are more recent than the respective .o files, **build** compiles the .o files according to the suffix rule, and puts the files in ../obj (the specified object directory).
3. Compares the modification times of member1.o member2.o member3.o.
4. If the modified times are more recent than **program**, **build** executes the action line that compiles the module with special ifdefs.

The command line:

build listing

shows that **build** is not necessarily restricted to compiling. This line prints module1.c module2.c and module3.c on the line printer if any changes occurred since the last *listing* file was made.

build performs the action line only if one or more of the following conditions are met:

- The module object does not exist as a real file
- Any of the objects upon which the module object depends are files with a more recent modification date than the program module.
- Any other action lines operate against any of the objects upon which the module depends.

DIAGNOSTICS

Object building will fail if:

- the building of an object upon which the object depends fail
- an action list ends because any command line returns an error
- **build** applies a suffix rule but the object is not found in the working directory when the actions complete
- the object cannot move into the object directory

build

build produces the following operation/error messages.
Explanations are provided where necessary:

Usage: **build** [-s] [mf=buildfilename] file1...filen\n
Cannot allocate memory for Buildfile (buildfile name)\n
No memory for > (n) Buildfiles
Bad flag: (a list of unrecognized flags)
Can't fork
 Can't fork process to execute command line
Can't exec /bin/command
 Can't exec shell to execute command line
Can't exec /bin/sh
 Can't exec shell to execute command line
code %d waiting for /bin/command
 Error occurred while waiting for child process
 to execute a command line

No mem

 A memory allocation failed

multiply defined (name)

 Name multiply defined

Too many \$includes

Can't \$include file named (name)

Name more than (n) chars long

\$include file missing

Can not open (name)

(name) does not say how to make (anothername)

(name) does not exist

Can't find any source for (name)

Recursion in dependencies of (name)

Couldn't make name

(name) is up to date

The following errors may occur while **build** tries to move object files to the object directory.

code %d waiting for /bin/sh

 Error occurred while waiting for child process
 to execute a command line

Can't add name (name) to (base name)

build was unable to create a name with **addname()**

Can't remove old name (name)

build

build tried to delete a file but failed
Can't open (name)
build tried but could not access a file
Can't create (name)
build couldn't create a file

ALSO SEE

nothing

NOTES

Because a new Command Interpreter runs for each command line, the **cwd** command is only in effect for the remainder of the command line.

USES

asm System assembler.

/bin/cc To do implied transition

/bin/command To execute command line

Buildfile To get dependencies when Buildfiles were unspecified

calltree

calltree generates a call tree or cross reference

FORMAT

calltree [*i=file*] [*r=routine*] [*options*] *file1..filen*

DESCRIPTION

calltree processes its input files and produces the static call structure of these files (intrepreted as a C program) as its standard output.

calltree detects any sequence of functions that call themselves. Therefore, it will not loop or expire in the case of recursion.

OPTIONS

- b Prints a vertical bar at each 'tab' stop for alignment purposes. See the -s options for adjusting the size of the 'tab'.
- c Inverts the structure of the tree. For each one defined, the list that follows is comprised of functions that directly call the function.
- f Produces a flattened tree of depth one (i.e., containing, at most, one instance of each function called from within the function).
- i=file* Causes another tree to be built out of the function names found in *file*. The file should contain only names; each name should be on a separate line. These names are then ignored when constructing the call tree. This option can be used to remove calls to standard library routines, or to remove entries about functions. Because multiple ignore files work correctly, libraries of such files can be created.
- l=N* Same as -o but the argument N (a number) is the maximum nesting depth (i.e., functions more than N calls from main do not get printed in the call tree of main).

calltree

- m Produces the call structure for main only. The default is to print out the separate call trees for every function.
- np Skips the preprocessor stage. Overrides the default -p option.
- o Stops the printing at the first level. Only functions called directly by this function are printed. -o is the same as l=1.
- p Causes the C preprocessor to be invoked on each of the input files before passing the files to the routine which constructs the tree. This is the default mode of operation.
- r=routine Produces the call structure for only the named routine. The default is all defined routines. Note that because this option is implemented as the general case of the -m option, only one of these two can be meaningfully specified.
- s=N Sets the indentation ('tab') value to N (a number). The default is 4. Very large projects might require a smaller value to prevent the output from creeping off the right side of the page. A value of zero prints the tree flat, although no elimination of duplicate entries occurs.
- u Lists all functions not called via main (i.e. all apparently unused functions). It should be noted that function names appearing only inside a structure (and presumably called only through that structure) will not be detected as callable from main. Also, the -u option includes the -f option by default; it is therefore redundant to specify both options.

EXAMPLES

The following command line produces a list for each defined function of all routines that use this function:

```
$calltree -c -f file
```

The following command line generates a list for each specified routine:

```
$calltree -c -f r=routine file
```

calltree

The following command line lists all functions that directly call the specified routine:

```
$calltree -c r=routine file
```

The following command line lists all functions that do not have the specified routine in their call trees (and therefore are immune to changes in the routine):

```
$calltree -c r=routine -u file
```

The following command line generates an alphabetical list of all the functions called by main:

```
$calltree -f -m file
```

The following command line prints the list of all functions called by the specified routine:

```
$calltree -f r=routine file
```

The following command line generates a flattened list of all functions that are not called from the specified routine:

```
$calltree r=routine -u file
```

DIAGNOSTICS

Close w/o open:

A negative brace count was encountered. The count was reset to zero. Presumably the program cannot be compiled in its present state.

Any other errors are fatal and the appropriate message is printed.

ALSO SEE

nothing

NOTES

For large projects, failing to invoke the *-m* option causes voluminous output which can easily exceed either the aggregate size of the source files or all available free space on the disk or both.

calltree

Constructions such as `typedef pointer (*funcs) ();` fool the program -- the function detector is not completely general.

USES

Any files used by the preprocessor.

casm

casm translates assembly language files to COFF object file

FORMAT

casm [-g][o=file][-s] file1...filen

DESCRIPTION

casm assembles the concatenation of the named files.

OPTIONS

-g Global

Treats all undefined symbols in the assembly as global (used by the C compiler).

o=*file* Output file.

Leaves the output of the assembly in the specified file, instead of in the default *t.out*.

-sShort.

Forces all absolute addresses to sixteen bits.

EXAMPLES

To put the output of the file *mch.m* into the file *mch.o* type

```
$casm o=mch.o mch.m
```

DIAGNOSTICS

The following is a list of the possible diagnostics:

Can not open <filename>

The assembler cannot open the source file.

Extra characters on line

Non-blank characters are found after the end of an assembler instruction.

Word at odd location

An attempt was made to assemble a word value at an odd

casm

location. This is probably due to a missing 'even' after a *.byte* or *.ascii* command.

bol> multiply defined

The <symbol> is defined twice with different values. This can also occur if the value of the <symbol> changes between the first and second passes of the assembler (phase error).

Bad assignment to '.'

An assignment is attempted, which would set '.' to a value smaller than the current location, or the value to be assigned is in a different segment (*.text* or *.data*) than the current one.

Missing op

An operator is missing in an expression.

<symbol> undefined

The <symbol> named is not defined in this file or is not marked as a global.

Operand type clash

An arithmetic operation is specified on operands of incompatible types.

Division by 0

Division by 0 is attempted in an expression.

Missing <char>

The specified character is expected but is not present. This is usually due to unbalanced parentheses.

Register required

A register name is required in the address mode.

Illegal type

An illegal address mode is specified for an instruction. Check the *68000 User's Manual* for legal addressing modes.

Wrong number of operands

The wrong number of operands is given for a 68000 instruction. The *68000 User's Manual* specifies the correct number of operands.

Too many operands

The number of operands given on the source line exceeds the maximum allowed by the assembler.

Bit number out of range

The bit number in a static bit instruction is outside the range 0-7.

Illegal addressing mode

An address register cannot be used as a byte-size operand.

Illegal size for memory shift

Memory shift instructions must be word-size.

Illegal trap value

Trap values must be in the range 0-15.

Symbol required

A symbol name is required, as in the *.global* operator.

Illegal string delimiter

An invalid delimiter is specified in an *.ascii* or *.asciz* command.

.ascii past end of line

The string in the *.ascii* or *.asciz* command continues past the end of the line. Check for a missing delimiter.

Absolute value required

A non-relocatable value is required.

Byte overflow

A value is specified that is too big for a byte.

Word overflow

A value is specified that is too big for a word.

Quick or count too large

The immediate value is too big for an *addq*, *subq* or shift instruction.

casm

ALSO SEE

ld binds COFF object modules
UNOS Developers Manual

NOTES

The assembled file is not normally executable. An executable file has no unresolved external references and no *.data* section in the program. The linker relocates references to the data section to a known address (0x400000) which the system sets up at run-time via the memory management system. (This address is hardware dependent.)

casm is a version of the **asm** program which generates COFF (Common Object File Format) files rather than traditional UNOS *.rel* files. Link these files with the **ld** linker, not **link**, and use them with the C compiler which generates COFF modules. While COFF does not restrict the length of identifiers, **casm** restricts identifiers to 80 characters, and the old C compiler and assembler restricted identifiers to 16.

USES

/sys/bctc "Big Convert To COFF" for output file conversion.

cat concatenates files

FORMAT

cat file1...filen

DESCRIPTION

cat opens each of its named files, reads all the characters it finds there, and writes them to standard output.

If **cat** has no arguments, then it reads from standard input. **cat** can be used to feed several files into a pipeline in a concatenated fashion.

Because **cat** simply dumps the contents of the file, it is the easiest way to convey a copy of a file to a device, such as the user's terminal or a floppy.

OPTIONS

none

EXAMPLES

To display the contents of a file on the terminal, type
\$cat file

To concatenate several files into one, type
\$cat file1 file2 >file3

DIAGNOSTICS

cat: Can't open <file>

ALSO SEE

nothing

cat

NOTES

Beware of this construct

```
$ cat *.doc > all.doc
```

cat creates *all.doc* before the **.doc* is expanded. *all.doc* becomes part of the input list, resulting in a loop that eventually uses up all the file space on the file system.

USES

nothing

cc compiles C programs

FORMAT

cc [option] ... file ...

DESCRIPTION

cc is the UNOS C compiler interface. cc coordinates the invocation of the C compiler, the assembler, and the linker.

When cc is given source files as arguments, it uses the file suffixes to determine what to do with them. Arguments whose names end with .c are taken to be C source programs. cc compiles, assembles, and links them. Arguments whose names end with .s are taken to be assembly source programs and are assembled, and linked. If a *filename* argument is specified without a suffix, cc treats it as a .c specification.

The following options are interpreted by cc. See ld for link editor options and ccom68 for compiler options.

- B*string* Constructs pathnames for substitute preprocessor, compiler, assembler, and link editor passes by concatenating *string* with the suffixes: cpp, c0, c1, or c2 (or optim). It uses the commands **as** and **ld** for this. If *string* is empty cc assumes the pathname is **/lib**.
- c Suppresses the link edit phase of the compilation and force an object file to be produced.
- C Includes comments in the preprocessor output. Only meaningful if **-E** or **-P** is specified.
- D*name* Defines *name* to the preprocessor with the value 1. This is equivalent to putting **#define name 1** at the top of the source file.
- D*name=string* Defines *name* to the preprocessor with the value *string*. Using this option is equivalent to putting **#define name string** at the top of the source file.

cc

- E Runs only the preprocessor (on the named C programs) and sends the result to the standard output.
- f Causes the compiler and linker to generate instructions for the SKY FFP floating point option.
- g Causes the compiler to generate additional debugging information needed for symbolic debuggers.
- help Issues the following information if it is the first flag, and is typed literally. A *-Wx,-help* flag is available, as listed later.

```
$ cc -help
usage: cc [ flags ] files...
Flags are:
-c          Compile to .o files
-o file     Linker puts output file in 'file'.
-f          Compile for SKY FFP
-g          Add line number information
-O          Add optimizations
-S          Compile to assembly files only
-Dname      C preprocessor: initially define 'name' to be '1'
-Dname=value Define 'name' to have given initial value.
-Uname      Override default definition of C preprocessor
            symbol 'name'.
-E          Preprocess only.
-P          Preprocess only to .i file
-C          When running preprocessor, do not strip comments
-Bstring    Find compiler programs with prefix
            'string' (/lib/o)
-t[cp012a1] Find only C compiler (cp012), assembler (a)
            or linker (l) with -B prefix
-WX,-arg1[, -args...] Send -arg1... to pass X
                    (pass 'x' is cc itself)
Other options will be passed to the 'ld' linker.
```

- i Has no effect but will not be rejected, for compatibility reasons.

- I*string* Searches for include file names (which are not absolute pathnames) in the directory *string* before searching the a standard list of directories. Multiple -I options can be specified. They will be searched in the order encountered.
- l*x* Search the archive library **lib*x*.a**, for *x* up to seven characters. Since libraries are searched in the order in which they appear in the command line, the placement of this switch is significant. The default areas for libraries are **/lib** and **/usr/lib**.
- L*dir* Searches for libraries in *dir* before looking in **/lib** or **/usr/lib** (the usual library locations).
- n Has no effect but will not be rejected, for compatibility reasons.
- o*outputfile* Tells the **cc** program the desired output file.

NOTE: If no *outputfile* is given to **cc**, it creates the output file name by stripping the suffix from the first **.c**, **.o**, or **.s** file given.

- O Causes loop optimizations. Must be passed using the **-Wc,-O** format (see NOTES section).
- S Compiles the named C programs and leaves the assembler-language output on corresponding files suffixed **.s**.
- t*[*p012alc*] Finds only the defined preprocessor, compiler, assembler, and link editor programs in the files whose names are constructed to a **-B** option. (These programs are sometimes called **cc** "passes".] In the absence of a **-B** option, the pathname is taken to be **/lib**. The value **-t""** is equivalent to **-tp012**.

NOTE: The C compiler does not separate passes **p**, **0**, **1**, or **2**. The passname **c** is equivalent to all of these.

- P Runs only the preprocessor on the named C programs and leaves the result on corresponding files suffixed **.i**.

cc

-Uname Undefines the predefined preprocessor symbol *name*. This is equivalent to putting `#undef name` at the top of the source file.

-Wx,arg1[,arg2...] Hands off the argument[s] to pass *x* where *x* is one of [0cal]:

0 (zero)

or *c* indicate the compiler

a indicates the assembler (ar command)

l indicates the link editor (ld command)

-Wx,arg1[,arg2...] is an extra UNOS flag that extends the C language standard to include arbitrary length variable names.

-Wx hands off the following UNOS flags to the compiler:

dbase=#

Sets the base address of the data segment.

-help

Issues the following help information IF it is the first modifier to **-Wx** and is typed literally:

```
$ cc -Wx,-help
```

```
usage of -Wx, flag:
```

```
tbase####      Sets text base address.
```

```
dbase####      Sets data base address.
```

```
sbase####      Sets stack base address.
```

```
ssize####      Sets initial stack size.
```

```
-probe         Enables stack probe instructions (defaults on,  
               -noprobe to disable).
```

```
-pop(alot)      Causes the compiler to clean up the stack in  
               loops to prevent excessive stack growth.
```

```
-(auto)reg      Causes the compiler to automatically assign register  
               variables (enabled by default, use -noreg to disable).
```

```
-sbt           Put stack before text for one-segment object file
```

```
-X###          Enable Greenhills C feature ###.
```

```
-Z###          Disble Greenhills C feature ###.
```

```
All flags must be separated by commas  
(e.g. -Wx,-sbt,ssize=0x8000)
```

-probe

(**-noprobe**)

Generates stack probe instruction "TSTB -256(SP)" at the beginning of function calls. *-probe* is the default, and *-noprobe* turns off this feature.

sbase=#

Sets the base address of the stack. The stack base, when set in this manner, must lie in the text or segments. The starting address of the stack (the top address) is *sbase* + *ssize*.

ssize=#

Sets the initial size of the stack segment. The default size is currently 4096 bytes.

-sbt

Sets the stack before the text. The text base is equal to *ssize*, and the text section will be non-sharable.

tbase=#

Sets the base address of the text segment.

-X###

-Z###

Passes those flags onto the Greenhills compiler. They may be specified under *-Wx* only.

Other arguments are taken to be either link editor option arguments or libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the name specified by *-o* or by the first filename given to **cc**.

RETURNS

cc returns a correct exit code (for the benefit of the **build** program). A 0 returns if it succeeds or an appropriate error code (whatever code is returned by one of the steps it execs).

EXAMPLES

file.o is the unlinked binary object file.

```
$cc -c file.c
```


cc

output to file.o The file becomes the executable program.
`$cc o=file file.c`

Assembly language is output to **file.s**.
`$cc -S file.c`

Preprocessor output is placed in **file.i**.
`$cc -P file.c`

DIAGNOSTICS

Self-explanatory

SEE ALSO

ld links editor for common object files

UNIX System Programmer Reference Manual

The C Programming Language by B. W. Kernighan.

Programming in C-A Tutorial by B. W. Kernighan.

C: A Reference Manual by Samuel P. Harbison and Guy L. Steele Jr.

NOTES

Charles River extends the C language standard to include arbitrary length variable names.

The following **ccom68** flags are on by default for the **cc** and **gcc** commands. Parts of UNOS code use the special **-U** and **-D** definitions.

-coff	-X102
-ga	-X122
-X12	-X138
-X14	-Uunix
-X16	-Dunos
-X22	-DM68000
-X81	-D___GREENHILLS
-X99	

See Appendix A of the *UNOS Developers Manual* for a table comparing the flags used by **gcc** and **cc**.

USES

/lib/cpp

/bin/ccom68

/bin/asm

/bin/casm

/bin/ld

/lib/c68rt0.o

/lib/libunos.a

ccom68 compiles C programs

FORMAT

ccom68 [options] file

DESCRIPTION

The **ccom68** program compiles C programs. It is usually called by either the **cc** or **gcc** command. Be extremely careful if you decide to use **ccom68** by itself.

The **ccom68** output is assembly language output, if you do not specify a **-E** option. Be aware of the following **ccom68** behaviors:

- An *.s* extension appends to each filename
- If the compiled file has a suffix *.c*, this suffix is exchanged by *.s*.
- A *.s* file is left in the same directory as the input file.

OPTIONS

- C** Includes comments in the preprocessor output. The default strips comments from the output.
- Dname** Defines *name* to the preprocessor with the value 1. This is equivalent to putting *#define name 1* at the top of the source file. (See some specific **-D** defaults in the NOTES section.)
- Dname=string** Defines *name* to the preprocessor with the value *string*. Using this option is equivalent to putting *#define name string* at the top of the source file.
- E** Does not compile the program. Places the output of the preprocessor on the standard output file. This option is useful for debugging preprocessor macros.
- f** Causes the compiler to generate instructions for the SKY FFP floating point option.

cocom68

- g Generate debugger information.
- ga Forces all routines to have frames. (cc default.)
- I*string* Searches for include file names which do not start with / in the directory *string* before a standard list of directories. Multiple -I options can be specified. They will be searched in the order encountered.
- O Causes loop optimizations. (See the NOTES section.)
- O2 Causes loop optimizations. (See the NOTES section.)
- U*name* Undefines the predefined preprocessor symbol *name*. This is equivalent to putting *#undef name* at the top of the source file.
- X9 Disables local (peephole) optimizer.
- X11 Generates a stack probe at the beginning of each routine which uses more than 40 bytes of stack space or which calls a routine.
- X12 Generates 68010 code rather than 68000 code. (cc default.)
- X13 Suppresses code generation and creates an output file of zero length.
- X14 Prepends an underscore to all programmer defined names. (cc default.)
- X16 Uses *.=+n* to allocate n bytes of blank space. The default is *.blkb*. (cc default.)
- X18 Does not allocate programmer-defined local variables to a register unless they are declared to be a register.
- X19 Does not set the high order bytes for functions which return less than a full word, e.g. *char x()*.

- X22 Uses IEEE format floating point constants and generates calls when the numbers change size. (cc default.)
- X23 Resets the stack increment before it can grow large.
- X25 Sizes the output routine on the error output as the compilation proceeds.
- X26 Separates I&D space and locates dispatch tables in data space. (MIT version)
- X29 Appends the suffix *.asm* to the output file (rather than *.s*).
- X30 Generates inline calls to the Sky floating point unit. Uses hardcoded constants for the device address.
- X35 Generates Motorola-style output code. This should be used with most cross development assemblers.
- X36 Presumes that all 'switch' or 'case' statements will take a defined label. This generates slightly faster and smaller code.
- X37 Issues warning when dead code is eliminated.
- X39 Does not move frequently used procedures or data addresses to a register.
- X42 Does not generate calls to single precision routines, except for conversions to and from double. Use this if you do not have single precision floating point arithmetic support.
- X55 Signs bit fields of type int, short, and char. The default is for all fields to be unsigned.
- X81 Allows initialization of external variables. (cc default.)
- X92 Generates format assembly code for a Berkeley version of the compiler. DO NOT USE with UNOS.
- X97 Does not generate the 'ascii' directive, use a sequence of bytes instead.

ccom68

- X98 DO NOT USE with UNOS.
- X99 Implements a CRDS flag. Currently, it allows redeclaration of a function's return type (generates warning). (**cc** default.)
- X101 Generates 68881 code. You must use -X22 with this switch. (See also -X122 and -X129.)
- X102 Instructs compiler to output assembler code appropriate for the Charles River **casm** command. (**cc** default.)
- X120 Performs the Hunter and Ready code translations on the assembly language code. Motorola format only.
- X121 Disables the machine-operand-specific portions of the local (peephole) optimizer. Use this switch or -X9 if the compiler seems to be incorrectly compiling a program.
- X122 Preassemble 68020 instructions, so that 68020 instructions can be processed by a 68000 assembler and linker. (**cc** default.)
- X129 Uses the 68881 instructions for evaluation of sin, cos, etc. This is done by intercepting the calls to these routines; you can still define 'sin,' etc., but the code will not call it.
- X130 Uses 68020 mode. Creates code that will only run on a 68020 system. It implies that floating point values come back in fp0, with 4-byte alignment.
- X133 Use the non-filling declaration DS.B rather than DCB.B in Motorola format. This prevents large arrays from taking up large amounts of output module size (and download time). However, they will not be initialized.
- X138 Turns off the UNIX 'lcomm' (local common) directive. (**cc** default.)
- X140 Uses 68020 alignment rules. This is a good idea if your C library has been recompiled with this switch. If you plan to upgrade to a 68020 system, it is a good idea to use this switch for ALL compilations.

- X164 Does not stop in the event of a code generator abort.
Occasionally useful in determining the cause of an abort.
- X168 Does not pull constant floating point expressions out of loops.
- Zxx Turns off switch xx. Useful if your version has a switch turned on by default. To generate code without stack probes, use: -Z11

EXAMPLES

This command line compiles file *x.c* producing assembly language output *x.s*. All routines of *x.s* will have frames, an underscore will be prepended to all programmer defined names and no registers will be allocated to programmer-defined local variables (unless they are declared register in *x.c*). The name *my_prog* will be defined to the preprocessor with the value 1.

```
$ ccom68 -ga -X14 -X18 -Dmy_prog x.c
```

DIAGNOSTICS

ccom68 produces self-explanatory diagnostics.

ALSO SEE

asm translates assembly language files

gcc compiles C programs using the Green Hills C compiler

wcc compiles C programs using the Whitesmiths C compiler

NOTES

The following **ccom68** flags are on by default for the **cc** and **gcc** commands. Parts of UNOS code use the special -U and -D definitions.

-ga	-X102
-X12	-X122
-X14	-X138
-X16	-Uunix
-X22	-Dunos

ccom68

-X81	-DM68000
-X99	-D__GREENHILLS

See Appendix A of the *UNOS Developers Manual* for a table comparing how **cc** and **gcc** use compiler options.

The **-O** and **-O2** options cause the compiler to try even harder to optimise loops. While Green Hills claims that these options never slow down a program, Charles River has found many cases where the **-O** option has a negative effect. We have found even more cases where **-O2** has a negative effect. If you have programs that use loops heavily, you might want to experiment with these two options. Sometimes they really speed things up.

The **-O** option can be passed by the **cc** driver only by using the **-Wc,-O** format.

USES

None

cd

cd changes working directory

FORMAT

cd directory

DESCRIPTION

cd changes your current directory to the specified directory. You can specify either a relative or an absolute pathname.

OPTIONS

none

EXAMPLES

To change to a specific directory, using the absolute pathname, type

```
$cd /usr/john/work
```

To change to a directory, using a partial pathname, type

```
$cd work
```

DIAGNOSTICS

none

ALSO SEE

pwd displays name of working directory

User's Guide to UNOS

NOTES

none

USES

nothing

cdeb debugs a program

FORMAT

```
cdeb [mlist=file][-nobj][p=pid][-unbuf] symfile
[datafile]
```

DESCRIPTION

cdeb lets you monitor the execution of a program and facilitate interactive debugging of COFF files. UNOS suspends processing when it discovers a fatal error. You can also suspend a process to debugging it and examine a suspended process.

cdeb takes symbol definitions from the *symfile*, and an optional *datafile* can supply the data. If you do not specify a *datafile*, **cdeb** uses *symfile* for both. The debugger understands the core file automatically. If you type

```
$cdeb filename
```

OPTIONS

Command Line Options

The **cdeb** command takes these options.

mlist=file Reads the file named for macro definitions. The file format has the same requirement for the command **%read**.

-nobj Examines all of the file. By default, **cdeb** accesses an object file by the address the object file would have when executed.

p=pid Examines the suspended process whose process id is given by **pid**.

-unbuf Opens the data file in the unbuffered mode. The buffered mode is the default mode. Although the buffered mode is more efficient, it prevents you from seeing the changes immediately. Use unbuffered when examining UNOS tables via **/dev/kmem** if you want to note changes as they occur.

cdeb

DEBUG COMMANDS FORMAT

The **cdeb** command format consists of an optional expression value followed by an optional command string and ending with a carriage return.

cdeb works by first reading in the program's symbol table and then prompting you with the special debugger prompt:

DBG>

Expressions

Expressions have values that you construct using the arithmetic operators +, -, *, and the operator @. The operator @ represents the value of the contents of the location. You can represent the value by *[size]* and *[radix]* specification. To avoid unexpected interpretations specify size and radix every time.

Every expression sets the current location. You can reference this location with the . (period) symbol.

The operands of an expression include

- numbers
- the symbol . (period) meaning the current position
- register names in parentheses (valid only if there is a process being debugged)
- names of external variables and functions (valid only if the symbol definitions are present in the symfile given on the command line)

Typical expressions are

```
main
.-20
4(fp)@lx
(pc)
```

Modifiers

You can use the following modifiers with some of the **cdeb** commands.

unsigned

This modifier specified by the character *u* prints the *[exp]* as an unsigned number. If you do not specify a *u* modifier, values shorter than 4 bytes are sign-extended before printing and the the decimal output is signed.

size modifier

This modifier indicates the size of the value. Use the following characters to specify the *[size]* modifier:

c char 1 byte
b byte
w word 2 bytes
s short 2 bytes
i int 4 bytes
l long 4 bytes

The default *[size]* is the last *[size]* specified in the process (initially *i*, int).

radix

This modifier specifies the printed format of the value.

d decimal
o octal
x hex

p pointer - the name of the nearest preceding external symbol and an offset (if not zero)

a ASCII

n do not display, but skip over the proper size item

z prints numbers until a 0 is reached, stops and does not display the last number.

? displays an instruction - ignores *[size]* and advances . (period) by the length of the instruction

The last *[radix]* specified is the default for this field (initially *d*, decimal).

Command Line

The four basic ways of combining *[exp][command string]* are

1. *[exp][command string]*
 Sets the value of the current location to the value of *[exp]*, and executes *[command string]*. The location begins at 0. Commands can alter the current location. Type .

cdeb

- (period) to display the current location.
2. `[command string]`
Resets the value of the current location to the value it had at the beginning of the last command, and executes `[command string]`.
3. `[exp]`
Sets the value of the current location to the value of `[exp]` and repeats the last `[command string]`.
4. `<cr>`
Leaves the value of the current location unchanged and repeats the last `[command string]`. For example, if the command is `?`, which displays an instruction, each `<cr>` will display the next instruction.

Command Strings

Multiple commands can appear on a single line and in any order. The only limitation is physical--255 characters.

Some commands accept an optional argument `<exp2>` before the command. Since the first `<exp>` on the line is assumed to be the starting position for the command, it is necessary to put a semicolon before `<exp2>` if it is the first expression on the command line. For example, to display twenty instructions starting at the same address as the last command:

```
DBG> ;20[?]
```

DEBUG COMMANDS

Commands

This section lists and describes commands of the debugger.

@i(:) Separates expressions and/or command strings in the command line.

`<=>` Prints either the value of `[exp]` or the current value.

string Prints the quoted string literally.

`/` Modifies the location expression or displays the current location.

Built-in Commands

A `%` character in the command string indicates a built-in

command except for the **!** command that is explained below.

! Passes the rest of the command line **string** to the command interpreter as defined by the environment variable *SHELL* (default is /bin/command) and waits for it to complete.

%breakpoint Sets a breakpoint at the location *[exp]* if specified, or at the current location.

%continue Continues a program for an active process from a breakpoint or suspension.

%define Defines a macro.

%delete Deletes a breakpoint at a specified location expression or at the current location.

%dloc Allows modification or view of memory.

%domain Sets the domain for routine **name** manipulation of local symbols.

%halt Returns to **cdeb** command level instead of continuing the process.

%kill Ends the current process (child) and returns to the debugger command level.

%list Lists the set breakpoints.

%loc Allows you to modify or view a specified local symbol. When you enter **%loc**, the most current call to the domain routine is found and calculates the locals symbol *[name]* address or register number.

%mlist Lists the specified macros and their definitions.

%newline Starts a new output line.

%pstep Executes similarly to **%step**. The difference between the two commands is in the execution. The **pstep** call is considered a one step command whereas **step** may stop at

cdeb

each routine until reaching the named routine.

%quit Leaves the debugger.

%read Reads in a file that contains macro definitions.

%registers Prints the current values of the registers for active processes.

%run Executes the program being debugged.

%stack Prints a stack trace for an active process.

%symbol Displays all the symbols in the symbol table. The symbol table shows the symbol name, the symbol type, TXT or BSS or DAT (see *UNOS Developers Manual*) and the symbol address. The symbol "G" terminator indicates a global.

%step Executes a single machine instruction. If an expression precedes the command, the command evaluates the expression then the program steps that number of instructions. A process must be active.

%undefine Removes the definition of the macro *macro_name*.

%write Writes all current macro definitions to a file *filename* in the format required by **%read** or the **mlist=filename** option.

Macros

The debugger provides facilities to create macros. You create macros by combining command strings up to 256 characters. Use the **%define** command or the **%read** command to define these strings. The *mlist=filename* option defines a string when you invoke **cdeb**. The **%write** command saves all macros used by **cdeb**.

When the debugger finds a \$ character in the command string it expects a macro name to follow. Then, the debugger interprets the command string associated with the macro name and executes it.

NOTE: You can nest macros but, nested macros can become recursive. Since there is no way to terminate the recursion, a loop results.

Iteration

The convention for specifying iteration is to bracket a command string and precede it with a count argument. Nest the commands within brackets.

Examples:

```
DBG>;10[/cdeb0ca/sd%newline]
        Prints 10 instances of a structure
```

```
DBG>;20[?]    Prints the next 20 instructions
```

Environments

If you want to save the current location pointer so you can return to it later, use { } to push and pop the current environment.

{ Pushes the current location pointer and sets the location to the argument provided.

} Pops the location to the saved value and restores the location pointer to the value at the last {, or at the beginning of the command.

; Restores the location but does not pop it.

For example, to print three words and the corresponding instruction:

```
DBG>{/cdebso}?
```

Notice how the { pushes the location so that later you can print the values in a different format. You can print unions in a similar manner. If a structure contains a union, the ; command prints it.

```
DBG>/21d {/1d;/sd;/10ca} .+10/sd
```

Since the ; command resets the pointer to the beginning of the union, you must use .+10 to advance to the next element of

cdeb

the structure. Step through the structure explicitly, because the { } combination has no effect on the location pointer.

```
DBG>/21d {/1d;/sd}/10ca /sd
OR
DBG>/21d {/1d;/sd;/10ca}/10cn /sd
```

EXAMPLES

```
$cdeb hello
```

```
DBG>!cat hello.c
```

```
main ().
```

```
{
    printf ("Hello, world.\n");
}
```

```
DBG>main/5?
```

```
main:
```

118	LINK	FP,#0xFFFC
122	TST.B	-256(SP)
126	MOVE.L	#0x40001C,(SP)
132	JSR	printf
138	UNLK	FP

```
DBG>printf%br
```

```
DBG>%run
```

```
Debugging process 662.
```

```
Process stopped at printf+4.
```

bpt/trace.

DBG>%sta

Process stopped at printf+4.

bpt/trace.

printf(0x400004)

main(0x1,0xDFFF6A,0xDFFFE0)

DBG>0x400004/zca

Hello, world.\012

DBG>%kill

Process 662 terminated.

DBG>%quit

\$

DIAGNOSTICS

none

ALSO SEE

asm translates assembly language files

ccom68 compiles C programs

gcc compiles C programs using the Green Hills C compiler

wcc compiles C programs using the Whitesmiths C compiler

UNOS System DEBUG Manual

NOTES

none

cdeb

USES
nothing

change

change substitutes text strings in files

FORMAT

change [-v] *from_string* *to_string* [*file1*...*filen*]

DESCRIPTION

change substitutes strings in files (as opposed to **translit** which substitutes individual characters).

change recognizes all the special characters of the pattern matcher. To specify a special character literally, you must precede it by a backslash \. When you use a special pattern matching character, quote the entire string to prevent the Command Interpreter from interpreting the string.

The character & in the *to_string*, replaces what was matched in the *from_string*. Multiple occurrences of & are allowed.

The change is made in place, that is, the file has the same name after the change, although the original file no longer exists. Change patterns are matched line by line, divide a two-line change into two single-line patterns.

Without any file names, **change** operates on standard input and writes to standard output.

OPTIONS

from_string Text you want to change.

-v Verbose. Displays lines before they are changed.

to_string New text you want *from_string* changed to.

EXAMPLES

To correct the spelling of "receive" in all "doc" files, type
\$change recieve receive *.doc

change

In the following command line, the dashes cause the equal sign to be interpreted literally:

```
$change "devspin" -- "dev=spin" chap.01
```

DIAGNOSTICS

filename: cannot open

ALSO SEE

User's Guide to UNOS

NOTES

If the pattern is not matched anywhere in the file, the file is not modified. Therefore programs which have not been modified do not need to be recompiled (useful to know if change was used for *.c files). The **build** command recognizes if files have not been changed.

USES

ch# In the same directory as the changed file.

chgpw

chgpw changes group password

FORMAT

chgpw [groupname]

DESCRIPTION

chgpw creates, changes or eliminates a group password.

When you execute **chgpw**, you are first prompted for the old password, and then twice for the new password (to double check that it's the password you want). The passwords (old and new) will not be echoed on the screen.

Only ADMIN can change a group's password.

OPTIONS

none

EXAMPLES

```
#chgpw sysadmin
Changing password for sysadmin
Enter current password:
Enter new password: Again:
```

DIAGNOSTICS

```
chgpw: Wrong password
chgpw: New password not entered the same way twice.
chgpw: Only admin can change a group's password.
chgpw: No entry in /etc/group for 'yepper'.
administration error: chgpw must run set-uid to admin.
```


chgpw

ALSO SEE

nothing

NOTES

When initially adding a password, type a <cr> in response to **enter current password:.**

USES

/etc/group checks the correct group password, then modifies **/etc/group** to contain the new password.

chgrp

chgrp changes the group associated with a file

FORMAT

chgrp group file1...filen

DESCRIPTION

chgrp changes the group-id of one or more files. Only the system administrator can change the group of a file.

OPTIONS

none

EXAMPLES

#chgrp other /etc /sys/unos

This command line changes the group of the directory */etc* and the file */sys/unos* to *other*.

DIAGNOSTICS

Can't change group of file.

User is not admin.

Can't access file.

File is inaccessible or non-existent.

No files given.

Command line error.

Bad group file.

Trouble with */etc/group*.

No group named group.

No such group is mentioned in */etc/group*.

No group or files given.

Command line error.

chgrp

Bad flag: flag

Command line error.

ALSO SEE

copy duplicates files

chmod sets or changes modes on files

chown changes owner of files

NOTES

none

USES

/etc/group to look up the group-id corresponding to the named group

chmod

chmod sets or changes modes on files

FORMAT

```
chmod symbolic_mode file1...filen  
chmod numeric_mode file1...filen
```

DESCRIPTION

chmod establishes or changes the access modes of the named files, to control access by the owner, the group, and the rest of the world. The modes may be specified as either an octal number (which specifies the new file modes), or a string of letters and operators (which can selectively change individual modes).

Only the owner or the administrator can change the modes of a file.

OPTIONS

Because the way UNOS looks at file permissions changed from UNOS Release 6 to UNOS Release 7, you may need to change the permissions on your files.

-groupfix option copies the world permission field to the group permission field. Type this command line to fix your permissions
`chmod -groupfix file1 file2 file...`

Symbolic modes are set by combining a class, an operator, and a permission in the form:

[class] operator permission [operator permission ...]

Several operators and permissions can be given for each class in question, and several symbolic modes may be given in sequence on a line by separating them with commas.

The class field is to be filled in with any combination of the following letters:

u user.

Indicates that the modes for the owner of the file is being changed.

chmod

g group.

Changes the group modes.

o other.

Changes the modes for the world.

a all.

changes the modes for all class. This is the same as if the class were omitted.

The operator is chosen from the following characters

+ adds a permission to a class

- takes a permission away from a class

=sets the permissions of a given class absolutely

The permissions may be any combination of letters chosen from the following:

r changes read permission.

w changes write permission.

x is execute permission for ordinary files, and search permission for directories.

s indicates that the file should set user or group id on execution. This only makes sense for classes *u* and *g*.

t non-implemented option

If the = operator is used, the permission field may be omitted to indicate that all permissions are to be removed for the selected class.

chmod

Alternatively, the modes of a file may be changed by specifying a numeric mode instead of a symbolic mode. The numeric mode consists of the sum of all the numbers associated with the modes to be selected for the file. These numbers are:

4000	Set user id on execution
2000	Set group id on execution
1000	Save text image (not implemented)
0400	read by owner
0200	write by owner
0100	search or execute by owner
0040	read by group
0020	write by group
0010	execute or search by group
0004	read by the world
0002	write by the world
0001	search or execute by the world

If it is desired that a file be inaccessible in all ways by all users, the mode **0** will eliminate all permissions.

EXAMPLES

To allow only write permission to the file `/doc/suggestions` by the file's own group and by the world, use the following command:

```
$chmod og=w /doc/suggestions
```

To set the file `/bin/login` to be set user id, type

```
#chmod u+s /bin/login
```

To change the modes of the directory `/tmp` such that it is completely accessible by everyone, use the following command:

```
$chmod 0777 /tmp
```

DIAGNOSTICS

Bad mode string

If the symbolic or numeric mode given is in error.

cannot change access modes of file

If the one who issues the command is neither the owner of the file nor the system administrator, or if a user who is not the

chmod

system administrator tries to set *t* permission.

cannot access file

If the file is inaccessible or non-existent.

ALSO SEE

chown changes owner of files

chgrp changes the group associated with a file

ls lists the contents of directories

umask displays the current file creation mode mask

NOTES

chmod creates all files to be 0666; if you desire them to be otherwise, use **chmod**.

USES

nothing

chown

chown changes owner of files

FORMAT

chown owner file1...filen

DESCRIPTION

chown changes the user-ID of the files. Only the system administrator, or a user with **admin** privileges, or the owner change the ownership of a file.

OPTIONS

none

EXAMPLES

To make the files **ps** and **cpw** owned by root, type
#chown admin /bin/ps /bin/cpw

DIAGNOSTICS

Self-explanatory.

ALSO SEE

copy duplicates files

NOTES

Some commands require root ownership to function properly (e.g., **login**, **ps**, and **cpw**).

USES

/etc/passwd To look up the *uid* corresponding to the named owner.

chron

chron performs periodic commands specified in file

FORMAT

chron [**-ec**]

DESCRIPTION

chron is run from **/etc/startup** for the purpose of invoking system programs at regular intervals. **chron** reads a table named **/lib/chronstab** to determine what intervals to run which commands. To change this table while **chron** is running, use any editor to change the file. If the flag **-ec** is not given, **chron** will detect that the file's modify date has changed and re-read it. If **-ec** is given, then advancing the eventcount **/lib/chronstab.ec** forces **chron** to re-read **/lib/chronstab** (which prevents **chron** from reading a half-written file).

The format of **/lib/chronstab** is an editable file containing one line for each system command to be run. Blank lines, and lines beginning with the character **#**, are ignored. Command lines have the following format:

MINS HOURS DWEEK MONTH MDAY command-line

The first five fields (separated by spaces or tabs) are given as ranges to match. The simplest ranges are **"*"**, meaning any time matching other fields; and a single number, indicating any time whose corresponding component matches that number. One can also specify a range of numbers as **#-#** (e.g., 10-15). Finally, one may specify ranges separated by commas, for example: 0,10-15,20,30,45-59. No spaces may occur within a single field. If any of the fields on a line do not appear to be legal ranges, the line will be ignored.

The ranges for each component are 0-59 for minutes and 0-23 hours, 0-7 for week day (1 is Monday, 0 or 7 is Sunday), 1-12 for month, 1-31 for day of month.

chron runs every minute, matching the current time to each command line range. For each command line, each component of the current time (minutes, hours, day-of-week, month,

chron

day-of-month) is compared to the corresponding range; if the time matches all five components, the command line (everything after the fifth time component) is given to **/bin/ksh** to be performed.

chron should be run as the administrator; however, **/lib/chrontab** should be protected against writing by unauthorized users for this reason.

Additionally, in the command line field, the character % is translated to a newline (unless protected by backslash (\)); only the first such line is taken as the command to execute; the rest is passed as the standard input to the command. A more general mechanism is to use IO redirection on the command line. Note that "command" error stream redirection (which is done with %) must be escaped as \%; to pass a \ character to "command" requires the sequence \\. A newline will be passed in the input stream at the end of the last line, so it is unnecessary to include a % at the end.

OPTIONS

-ec check **/lib/chrontab.ec** for modifications rather than modify date of **/lib/chrontab**.

EXAMPLES

Here is a sample *chrontab* file:

```
$cat chrontab
# the next line increments an event count every minute,
# every time chron runs
* * * * * advance /lib/minutes_up.ec
# every hour, let us increment an hour count
0 * * * * advance /lib/hours_up.ec
# for triskadecaphobes:
0 0 5 * 13 wall "IT'S MIDNIGHT, FRIDAY THE THIRTEENTH!"
# for multi-user systems
3 0 * * * diskusage /usr/* >> /etc/disk_hogs
# and perform "at" requests periodically
0,15,30,45 * * * * /lib/atrun
# example of multi-line commands
* * * * * cat > /testfile %Demo multi-%line commands.
```

chron

DIAGNOSTICS

none

ALSO SEE

when executes a command line at a given time

whenq inspects queue of **when** requests

whenrun runs queued **when** requests

NOTES

none

USES

/lib/chronstab Table of requests

/lib/chronstab.ec Eventcount for changes

cofftorel

cofftorel converts a COFF format object file to the old UNOS object file format

FORMAT

cofftorel [-h] coff_file rel_file

DESCRIPTION

cofftorel converts COFF files to the object file format.

OPTIONS

none

EXAMPLES

Convert the COFF object file, *COFF.o*, to the old UNOS object format and place the results in *HACK.j*.

```
$ cofftorel COFF.o HACK.j
```

DIAGNOSTICS

Self-explanatory

ALSO SEE

cofftorel converts a COFF format object file to the old UNOS object file format

NOTES

none

USES

nothing

command

command executes a command line or file

FORMAT

command [options][name [arg1...]]

DESCRIPTION

command reads and supervises the execution of most user command lines. **command** recognizes macros, special characters, control pipes, I/O redirection and detached processes.

You can use **command** to interpret files of commands. (See examples)

OPTIONS

- c Specified command line as an argument. Programs that execute command lines use -c.
- ce Provided for use with the UNIX **make** command.
- e Provided for use with the UNIX **make** command.
- i Force interactive prompting. (-i is assumed if standardin is a terminal.
- v Verbose mode. Echoes command before its execution.)

command

SPECIAL CHARACTERS

This list represents special characters recognized by the Command Interpreter as pattern matching characters or control information.

<u>Character</u>	<u>Meaning</u>	<u>Example</u>
!	OR	<code>\$list this!that!the foo</code> Lists "this", "that", or "the" in foo.
*	any number of characters (or none)	<code>\$list *</code> Lists all files except those that begin with . (dots), such as .EDTMP.
?	any one character	<code>W?</code> matches "W1", "Wh", etc.
{ }	precedence grouping	<code>\$list {foo!bar}*{c!bak}</code> Lists any files that begin with "foo" or "bar" and end with "c" or "bak"; could list "food.bak" or "barc".
#c	any number of occurrences of the following item	<code>#a</code> matches "a", "aa", "aaa", and so on. <code>#?</code> is the same as *
#	indicates special Command Interpreter commands	<code>\$#x</code> displays your initial environment
[-]	delimit a set of ASCII characters	<code>\$match "[A-Z]*" foo</code> Finds strings in foo that start with a capital letter.
^	beginning of line -- when used as a command argument	<code>\$change "^ " "+" foo</code> Substitutes a plus sign for a blank space at the beginning of a line in foo.

command

or

	not -- when in brackets	<code>\$list [\^a-z]*</code> Lists everything that does not begin with a-z.
<code>\</code>	escape a character from the pattern matcher	Type <code>\\</code> when you want a literal <code>\</code> . If you want to escape a control character (<code>^D</code> , or <code>^Y</code>) so that it won't be executed, use the Escape key. <code>\$ p \[*</code> Displays the file <code>[*</code>
<code>\$</code>	end of line This only works as a command argument, not with the Command Interpreter.	<code>\$change "file\$" "files" tmp</code> This changes the word 'file' that appears at the end of the line in the file tmp .

These special characters direct command action.

<code>%</code>	no characters -- when used as a command argument	<code>\$match "foo{.c!.j!%}"</code> Finds <code>foo.c</code> , <code>foo.j</code> , and <code>foo</code> .
<code>></code>	directs to output file	<code>\$list [a-b]* > newfile</code> Lists files beginning with <code>a</code> and <code>b</code> , and puts the list into <code>newfile</code> .
<code>>></code>	directs and appends standardout output file	<code>\$list newfile >></code>
<code><</code>	specifies input file	<code>\$count -l < list.file</code> Counts lines in a file. Reads from <code>list.file</code>

command

%	specifies standard error.	<code>\$/sys/daemon printer %\ /tmp/printer.errlog</code> Starts daemon and puts errors in printer.errlog file.
%%	directs and appends standardout to output file	
;	ends one command, begins another	<code>\$list;</code> delete newfile newfile lists and then deletes
	uses the output of the command on the left input to the command on the right	<code>\$list a* count</code> as Counts the number of files beginning with a, then list those files.
&	places a command in the background	<code>\$/sys/daemon printer&</code> Starts the printer daemon and places it in background.
&&	indicates a Boolean AND	<code>\$cc foo.c && delete foo.j</code> Deletes only if the cc executes successfully.
	indicates a Boolean OR	Equals a ;.
()	groups commands	<code>\$(match bob /doc/subs/=; echo done)&</code> Both the match and echo commands are done in the background sequentially. Executes commands with () from single shell commands.
@	represents a comment character, all other following characters are ignored.	<code>\$list *</code> This is a comment. Useful in script files.

EXAMPLES

`$command file arg1...argn`

When a file consists of a series of commands, **command** reads the commands sequentially, then executes them. You can substitute the arguments for `$1`, `$2`, etc., in the command lines, just as in the **do** command.

DIAGNOSTICS

command: File does not exist. Can't open filename

When the file specified cannot be located.

ALSO SEE

performs special Command Interpreter functions

User's Guide to UNOS

NOTES

command notifies the user of killed, suspended or otherwise unusually terminated processes. Notification occurs while

command waits for another command to complete or when the user types wait.

USES

/etc/passwd To determine the user's login directory

.globals For the user's defined global macros

locals In the current directory for local macro definitions

init Executes as a script file on initial startup

common

common finds identical lines in two different files

FORMAT

`common [-1][-2][-b] file1 file2`

DESCRIPTION

common finds lines that are either common between two files or exist in one file but not the other. The input files must be sorted.

OPTIONS

- `-1` Shows only lines that exist in the first file but not the second.
- `-2` Shows only lines that exist in the second file but not the first.)
- `-b` Shows only common lines in both files.

EXAMPLES

To display entires in the file *words* that are not in the file *dictionary*, type
`$common -1 words dictionary`

DIAGNOSTICS

none

ALSO SEE

compare compares two binary files

difference indicates how two text files differ

NOTES

If only one file is given as an option, **common** assumes standard input for *file1*.

common

common counts white space and tabs when comparing files, so two seemingly identical files might appear quite disparate.

USES

nothing

compare

compare compares two binary files

FORMAT

`compare [-all][begin=#][count=#][-s] file1 [file2]`

DESCRIPTION

compare examines one file and standard out, or two files on a byte by byte basis, and prints the file position of the first difference it finds, first in decimal and then in hexadecimal. Its principal use is in determining whether two object files are identical.

OPTIONS

Note all options can be abbreviated to their first letter.

-all Compares all bytes in the range, or else stops on the first error.

begin=# Begins the comparison at byte n, otherwise at byte 0.

count=# Stops after n comparisons, otherwise at the first EOF or first error if *-all* is not used.

-silent Does not print messages, just returns the exit code.

EXAMPLES

To compare two files on a byte by byte basis, type
`$compare basic /bin/basic`

The following example executes **notify** which then executes **compare** in the background. **notify** also creates a file and directs the output of compare to that file (see the **notify** command):

`$notify compare basic /bin/basic`

compare

DIAGNOSTICS

When you use **compare** with **notify** (as in the second example, above), the exit codes are displayed

- 0** files are the same.
- 1** file 1 is longer.
- 2** file 2 is longer.
- 3** files differ before EOF is reached on either.
- 4** wrong number of arguments.
- 5** cannot open one of the files.
- 6** I/O error occurred on one of the files.

If you use **compare** alone (as in the first example, above), the messages are self-explanatory.

ALSO SEE

difference indicates how two text files differ
common finds identical lines in two different files

NOTES

compare cannot be used to determine if a recompiled program has the same instructions as an older program, because even though the two files might be the same, the version dates will always be different.

USES

nothing

confedit creates and/or edits a UNOS configuration file

FORMAT

`confedit [-help] [-batch] [-verbose] [<file_name>
file_name>`

DESCRIPTION

confedit looks for the configuration file named. If the file exists, **confedit** opens and checks it for configuration file format. An error is reported if the file is not in the correct format. If the file does not exist, it is created. Assignment statements are read from stdin in the non-interactive mode and parameters change accordingly. In the interactive mode (the default) the parameters are prompted one at a time so you can enter the desired values. You must be signed on as system administrator (with the **ADMIN** prompt) to use the **confedit** command.

OPTIONS

interactive prompts for each parameter in sequence. You can recover format errors.

Conventions for interactive mode follow. In the following list, */n* stands for carriage return.

\n leaves the parameter unchanged and goes to the next parameter

v\n changes the current parameter to *v* and goes to the next parameter

^\n leaves the parameter unchanged and goes to the previous parameter

-n also leaves the parameter unchanged and goes to the previous parameter

?n displays the help text for the current parameter, if available

confedit

help\n displays the help text for the current parameter, if available

done\n exits and updates the file

The acceptable format is the same as that displayed as the current value. [*text*] is a string, enter "text"

[*n, n*] is a device specifier, enter n,n

[*n*] is a number, enter n

[*\oct*] is an octal char, enter '\Onn' or '\Onnn.'

-batch allows you to create a text file with the desired changes to the configuration file(s).

-verbose displays the old and new parameter values as they change.

-help displays the usage for **confedit**.

If you enter parameters that do not correspond to the correct form, a **Format error**. displays and the field reappears. The parameter is unchanged after syntax and/or format errors.

EXAMPLES

The following example displays the default mode, interactive mode.

```
confedit conf
ROOT device (0, 0 means ask). [4, 10] = *use default
Swap device (NULL means ask). ["/dev/swap"] = \
"/dev/swap2" *change
Swap device size in blocks. [8000] = ^ *back to prev
Swap device (NULL means ask). ["/dev/swap2"] = *no change
Swap device size in blocks. [8000] = 8000 *change
Number of system buffers. [80] = 80 *etc.
Number of process control blocks. [100] = 100
Ceiling priority for start process. [100] = 100
Floor priority for start process. [32] = . *non-numeric
```

```

Format error.          *error
Floor priority for start process. [32] = 32 *retype
Priority for swapper process. [200] = 200 *etc.
Number of logical units for whole system. [400] =
Number of open files for whole system. [200] =
Number of event calls for whole system. [100] =
Number of (shared) data segments. [200] =
Priority tuning interaction credit. [512] =
Priority tuning quantum used up demerit. [8] =
Bytes of input buffer per tty. [256] =
Maximum mounted file systems. [8] =
Size (in bytes) of histogram log area. [2000] =
Clock chip input rate in Hertz. [1250000] =
Should wait before mounting root? (y/n). [No] =
Maximum number of file locks (# locklists). [40] =
Default erase character. ['\177'] =
Default line kill character. ['\25'] =
Name of computer system. ["" ] =
Blocks in multi block I/O buffer. [5] =
Bytes of system error log buffer. [1024] =
Recovery ROOT ('0, 0' means ask). [4, 8] =
Name of script. ["" ] =
Auto start? (y/n). [Yes] =
Mount ROOT read-only? (i.e. tape) (y/n). [No] =
Number of pseudo teletype devices. [2] =
Poll after this many services. [20] =
Amount of memory to grab for distributed systems \
structures. [131072] =
Number of ipc queues. [0] =
Total number of message bytes. [0] =
Maximum size of a single ipc message. [0] =
Maximum message bytes on one ipc queue. [0] =
All created files are contiguous? (y/n). [No] =
Allocation chunk size in blocks. [0] =
Minimum allocation size for contiguous files in blocks. [0] =
Maximum allocation size for contiguous files in blocks. [0] =
Disable outgoing Distributed file system requests? (y/n). [No] =
ROOT device (0, 0 means ask). [4, 10] *wrap to 1st
Swap device (NULL means ask). ["/dev/swap"] = done *exit

```

confedit

To create a batch mode input file, create a file that includes the parameters you want changed in the configuration file. Include only those parameters that will change. For instance, make a file called *cmds* that changes the *cnf_start_ceil_pri* parameter to 50 and the *cnf_ttybufsize* to 512.

With this command line

```
#confedit -b <cmds conf
```

the *conf* file changes according to the instruction in the *cmds* file.

The format of the input file is

structure_name=parameter

structure_name is taken from */include/sys/conf.h* and parameter must be in the same format as described earlier.

For example, the following input file

```
confedit -b foo < bar
```

uses the following file *bar*

```
cnf_rootdev=4,10 cnf_swapdev[32]="/dev/swap2"
```

```
cnf_swapsz=8000
```

This will change the configuration file *foo*. The root device, swap device, and swap device size will all be modified. All other fields remain untouched.

DIAGNOSTICS

Format error.

is issued whenever your input makes no sense in the current context.

Syntax error.

is issued in non-interactive mode when a parameter name is not recognizable.

Non unique abbreviation.

is issued in non interactive mode whenever a parameter name is not unique.

confedit

ALSO SEE

confable displays either the contents of the current running configuration table or the contents from a configuration file

NOTES

No checking is done on strings for illegal characters.

No range checking is done for number parameters.

USES

none

conftable

conftable displays either the contents of the current running configuration table or the contents from a configuration file

FORMAT

```
conftable [-c!current] [-h!help] [file_name] ...  
[file_name]
```

DESCRIPTION

conftable reads from a file, a group of files or from the current memory resident configuration table and displays the information in a list to standard out.

OPTIONS

-c,current displays memory resident configuration table

-h,help displays the usage for conftable

file_name names the file to display.

EXAMPLES

This example explains how to display the configuration from the file *conf*.

```
$conftable conf
```

Configuration from file 'conf'

```
ROOT device is    4, 10.  
Swap device is '/dev/swap'.  
Swap area is 8000 blocks.  
80 system buffers allocated.  
100 process control blocks allocated.  
Ceiling priority is 100 for start process.  
Floor priority is 32 for start process.  
Swapper priority is 200.  
Maximum logical units is 400.
```


conftable

Maximum number of open files is 200.
Maximum number of event calls is 100.
Maximum number of shared data segments is 200.
Priority tuning interaction credit is 512.
Priority tuning disk wait credit is 0.
Priority tuning quantum used up demerit is 8.
Input buffers are 256 bytes per tty.
Maximum mounted file systems is 8.
Histogram log area is 2000 bytes.
Clock chip rate is 1250000 Hertz.
Will not wait before mounting ROOT.
Maximum number of locks (lock lists) is 40.
Default erase character is '\177'.
Default line kill character is '\25'.
System name is ''.
5 blocks in multi block I/O buffer.
1024 bytes in system error log buffer.
Recovery ROOT device is 4, 8.
Script is '/etc/startup'.
Auto start.
Mount ROOT read/write.
Number of pseudo teletype devices is 2.
Poll after 20 services.
Distributed systems structures memory is 131072.
Number of ipc queues is 0.
Total number of ipc message bytes is 0.
Maximum size of a single ipc message is 0.
Maximum ipc message bytes on one queue is 0.
All created files are not contiguous.
Non contiguous file allocation chunk size is 0 blocks.
Default minimum contiguous chunk size is 0 blocks.
Default maximum contiguous chunk size is 0 blocks.
Outgoing distributed file system requests are enabled.

End of Configuration 'conf'

This example explains how to use **conftable** to find what the current configuration is on your system.

```
$conftable -c
```

Current configuration from Memory

ROOT device is 4, 10.
Swap device is '/dev/swap'.
Swap area is 8000 blocks.
80 system buffers allocated.
100 process control blocks allocated.
Ceiling priority is 100 for start process.
Floor priority is 32 for start process.
Swapper priority is 200.
Maximum logical units is 400.
Maximum number of open files is 200.
Maximum number of event calls is 100.
Maximum number of shared data segments is 200.
Priority tuning interaction credit is 512.
Priority tuning disk wait credit is 0.
Priority tuning quantum used up demerit is 8.
Input buffers are 256 bytes per tty.
Maximum mounted file systems is 8.
Histogram log area is 2000 bytes.
Clock chip rate is 1250000 Hertz.
Will not wait before mounting ROOT.
Maximum number of locks (lock lists) is 40.
Default erase character is '\177'.
Default line kill character is '\25'.
System name is ''.
5 blocks in multi block I/O buffer.
1024 bytes in system error log buffer.
Recovery ROOT device is 4, 8.
Script is '/etc/startup'.
Auto start.
Mount ROOT read/write.
Number of pseudo teletype devices is 2.
Poll after 20 services.
Distributed systems structures memory is 131072.
Number of ipc queues is 0.
Total number of ipc message bytes is 0.
Maximum size of a single ipc message is 0.
Maximum ipc message bytes on one queue is 0.
All created files are not contiguous.

conftable

Non contiguous file allocation chunk size is 0 blocks.
Default minimum contiguous chunk size is 0 blocks.
Default maximum contiguous chunk size is 0 blocks.
Outgoing distributed file system requests are enabled.

End of Configuration from Memory

DIAGNOSTICS

If you name a file that is an illegal configuration file, the message ***** NOT A CONFIGURATION FILE ***** displays.

ALSO SEE

nothing

NOTES

none

USES

nothing

contig make a contiguous file

FORMAT

```
contig [extent=range] [alloc=rangelist] [-freeze]
[-truncate] file ...
```

DESCRIPTION

contig makes a contiguous file (a file which owns disk space in large extents rather than in single blocks). If the file named already exists, it must be truncated in order to become contiguous; this command saves the old contents of the file (in a file in **/tmp** or in main memory, depending on size) and writes it back later.

Contiguous files have two parameters, *min* and *max*, which govern allocation of disk space as the file grows. Whenever a write past the end of the space currently used by the file is done, the file system allocates a new extent of at least *min* blocks and at most *max* blocks (or at most large enough to contain the write, if that's larger). The write fails if there is no free extent of at least *min* blocks. There are system default values for *min* and *max*, which are used if the file's *min* and *max* are -1. All allocation is inhibited when the *max* value is 0.

OPTIONS

extent=range Sets the file's *min* and *max* allocation parameters, but does not cause any space to be allocated. *range* is a number (sets both *min* and *max*) or a pair of numbers separated by '-' (e.g. *extent=1024-4096* sets *min* to 1024 and *max* to 4096). The number(s) can end in 'b' or 'k' to indicate units of blocks (times 512) and kilobytes (times 1024). This multiplier will be applied to both *min* and *max* even if only *max* specifies it, so *extent=10-20b* means *extent=10b-20b*. If *extent=* is omitted, *min* and *max* are set to -1, which causes the system default values to be used.

alloc=rangelist Allocates one or more extents of disk space to the file. *rangelist* is a range (as above) or a list of ranges

contig

separated by ',' (e.g., *alloc=10b,10b* causes the file to have two ten block extents). If *alloc=* is omitted, *contig* tries to get a single extent to hold the current file contents (if any) by assuming *alloc=512-filesize*.

-freeze Marks the file to prevent any further allocation, by setting *extent* to 0. Any attempt to write past the end of the allocated space will fail. Since the *filesize* may indicate the file ends anywhere in the last extent, it is still possible to extend the file up to the allocation limit. The **contig** command always unfreezes files if *-freeze* is not specified. *-freeze* and *extent=* are mutually exclusive.

-truncate Discards any current file contents and disk space. Truncating a file requires write access to it.

EXAMPLES

To turn all commands into contiguous files for speed:

```
$contig /bin/* /sys/* /usr/bin/*
```

To prevent additional allocation of space to a contiguous file:

```
$contig -freeze able
```

To allocate 5 1000 block extents:

```
$contig -truncate alloc=1000b,1000b,1000b,1000b,1000b  
able
```

To cause subsequent allocations to be 300 blocks while allowing allocations down to 100 blocks to be acceptable if 300 blocks is not available:

```
$contig extent=100-300b able
```

DIAGNOSTICS

none

ALSO SEE

nothing

NOTES

-freeze does not prevent the file from being truncated, for example by `> filename` in the shell, or the **create()** subroutine. If a frozen file is truncated it will be frozen at zero length, so no write to it can succeed.

USES

nothing

copy

copy duplicates files

FORMAT

copy [options] [from_files] [to_file!directory]

DESCRIPTION

copy copies files.

If more than one file is specified on the command line and the last one is a directory, all previously specified files are copied into that directory. This is true for all types of files -- directories, devices, eventcounts, named pipes, and source files.

If the last file on the command line is *not* a directory, specifying more than two files causes an error. The first file is the source and second the target.

- If the second file exists, a copy of the first file is written into the second, preserving all links to the second file.
- If the second does not exist, it is created and a copy of the first file is written into the second.

Unless you specify the *-interactive* option with less than two file names, the command line is rejected as syntactically invalid.

You must have write permission to the destination file, if it already exists, and to the directory to which you are writing. The *-force* option (described below) lets you copy to a file even if you don't have write permission. (But you must have write permission to the directory)

If the *from_file* is a directory, **copy** creates not only the target directory, but recursively copies all subfiles.

copy

When the command is done, you'll see an exit message that indicates what was accomplished:

- If no files are copied, you'll see the message:
No files copied.
- If one file was copied, you'll see the message:
One file copied.
- If two or more files were copied, you'll see the message:
2 files copied
(or 3..4..5...etc)
- If one file was not copied, you'll see the message:
One file not copied.
- If two or more files were not copied, you'll see the message:
2 files not copied.
- If some files were copied and others were not, you'll get two messages telling how many were copied and how many were not.

Use the *-silent* flag to avoid printing messages to standard out.

OPTIONS

All options can be abbreviated to the first letter except *-setowner* and *-settime*.

- force* Use this option to copy to files for which you do not have write permission. You must have write permission to the directory.
- help* Prints, to standard error, a few lines listing all the command options **copy** accepts.
- interactive* Prompts the user for more files after copying any files given on the command line. You are first prompted for the file to copy from, and then for the file to copy to. Type a ^D or press the RETURN key on a blank line to end the command.
- merge* Prevents files with duplicate names from being overwritten.
- query* Asks whether a file should be copied. A response of either **yes** or **y** copies the file; any other response is

copy

interpreted as negative.

- recursive* Defaults to recursive behavior. To prevent recursive behavior use the options *-nor* or *-norecursive* on the command line. Source directories on the command line are ignored if recursion is prevented.
- silent* Prints no messages to standard out. No exit message is printed.
- setowner* Attempts to copy the original user id. *lm10 -settime* Duplicates the modification time of the source file for the destination file.
- verbose* Lists files that have been copied.

EXAMPLES

Create two files in the same directory with the same contents, but with different names:

```
$copy file1 newfilename
```

Create a file with the same name in a specific directory: **\$copy file1 directory2**

Use the **copy** command and get a prompt for source and destination:

```
$copy -i  
Copy from:  
Copy to:
```

DIAGNOSTICS

Last file <filename> does not exist.

If more than two files are specified on the command line, the last file must exist.

Last file <filename> is not a directory.

If more than two files are specified on the command line, the last file must also be a directory.

copy

Could not open <filename>.

File to be copied could not be opened.

Write access denied to <filename>.

The destination file does not have write access and the *-force* option was not used.

Could not delete <filename>

Could not delete the old file, before creating the file to copy into. If you are copying into an existing file of a dissimilar type, an attempt is made to delete the "to" file.

Can not make directory <filename>

Could not create directory after deleting the old file.

Cannot read from <filename>

Could not read from the file to be copied.

Cannot copy <filename> to itself <filename>

You cannot copy a file to itself.

Overwrite not allowed of <filename> with <filename>

If *-merge* option is used **copy** will not overwrite a file with the same name.

Cannot create named pipe <pipename>

If copying a pipe, could not create a new pipe.

Cannot open <filename> for writing

Could not open the destination file to allow write access.

Could not make event <eventname>

If copying an eventcount, could not create a new eventcount.

Could not make device <devicename>

If copying a device, could not create a new device.

Unable to set access modes of <filename>

Could not set the access modes of the destination file to be that of the **from_file**. The file is already copied.

copy

Unable to set owner of <filename>

-setowner option was specified in the command line but was not able to change the owner. The file is already copied.

Could not set modified time of <filename>

-settime option was specified in the command line but was not able to set the time as requested. The file is already copied.

Unable to open newly created file <filename> Destination file created but inaccessible.

ALSO SEE

addname gives a file another name

move renames files or moves them across file systems

NOTES

Whenever possible, **copy** will retain the original user access modes.

Copying a device file results in only copying the device file entry, not in reading the device.

When you copy eventcounts, if the target eventcount does not exist, the values of the user eventcounts are set to 0. If the target eventcount does exist, the copy command does not change the eventcount values.

If you use the **-f**, force option, then forcibly copied files will lose their pre-existing links.

USES

nothing

copy_boot

copy_boot copies a bootstrap program to a device

FORMAT

/sys/copy_boot bootprogram device

DESCRIPTION

copy_boot initializes a disk with information so that it can be booted. It copies the specified program to the disk, removing the object header, starting after the bad block map (if any) usually at block three (offset 0x600 from the start of the device) and writes in block 0 a volume ID containing the address and length of the program.

If the boot program uses less than 0x1200 bytes (occupies 6 or fewer 512 byte disk blocks) for text and data combined, it will fit along with the PVII and the bad block map in the space allocated before the default beginning of partition 0 on a hard disk (block 12). On a floppy disk (which has no partitions or bad block map) it will fit before the default beginning of the filesystem header (block 9).

If **copy_boot** finds that the boot program would overwrite the file system header and thereby destroy the file system it will ask if the you really want to do this. If you overwrite a filesystem header, you must use **makefs** to create a new filesystem on the disk. **makefs** automatically puts the filesystem header after the boot program if it finds one.

The boot program can be loaded and run from UNBUG. It loads at location 0. Execution begins by simulating a RESET -- SP (Stack Pointer) gets location 0 and PC (Program Counter) gets location 4.

A standard boot program called **/sys/unboot** is provided with the UNOS release. It can boot from any boot device currently supported by UNOS.

copy_boot

OPTIONS

none

EXAMPLES

To make a minimal bootable UNOS floppy:

```
$/sys/copy_boot /sys/unboot /dev/rcfd0
$makefs /dev/cfd0 1989
$mount /dev/cfd0 /mnt
$makedir /mnt/bin /mnt/etc /mnt/sys /mnt/dev
$copy /bin/command /mnt/bin
$copy /sys/unos!iop /mnt/sys
$copy /etc/start /mnt/etc
$copy /dev/tty0 /mnt/dev
$unmount /mnt
```

DIAGNOSTICS

Self-explanatory error messages are produced if either the boot program or the device is inaccessible or non-existent and if the boot program does not have a valid object header.

ALSO SEE

CP-32 USER'S MANUAL

makefs initializes a file system

NOTES

none

USES

nothing

count

count counts items in files

FORMAT

count [-chars][-words][-lines][-total] file1...n

DESCRIPTION

count reads all of the specified files and produces one line of output for each input file. This output consists of:

- the number of the characters
- the number of 'words'
- the number of lines in the specified files

Words is an arbitrary term in reference to programs, because they tend to have a lot of punctuation. For text files, a word is a sequence of characters separated by white space.

If more than one file is given, a grand total is also printed. If no filenames are given, **count** reads from standard input.

OPTIONS

All options can be abbreviated to their first letter. Any combination of options works; the default is (-c -w -l -t).

- chars* Counts characters.
- words* Counts words.
- lines* Counts lines.
- total* Grand totals of several files.

EXAMPLES

To count the number of characters in a file, type
\$count -c file
file: 128 chars

To count characters, words, and lines in several files, type
\$count file.*
file.1: 123 chars, 115 words, 6 lines
file.2: 265 chars, 467 words, 20 lines
total 388 chars, 582 words, 26 lines

count

To ascertain the number of files in the current directory, type
`$list | count -lines`

DIAGNOSTICS

"count: cannot open <filename>

ALSO SEE

nothing

NOTES

none

USES

nothing

cpw changes password

FORMAT

cpw [username]

DESCRIPTION

cpw creates, changes or eliminates your password.

When you execute **cpw**, you are first prompted for your old password, and then twice for your new password (to double check that it's the password you want). Your passwords (old and new) will not be echoed on the screen.

Admin may change the password of another user by giving the name of that user on the command line. The procedure is the same as above.

OPTIONS

none

EXAMPLES

#cpw jeff

Changing password for jeff.

enter current password:

enter new password: again:

#cpw Changing password for admin.

enter current password:

enter new password:

again:

DIAGNOSTICS

Self explanatory.

cpw

ALSO SEE

chgpw changes group password

login verifies user ID and password, and starts top-level shell

NOTES

If you are a new user without a password, when you initially log on to the system press the RETURN key in response to the **Password:** prompt. Then use the **cpw** command to give yourself a password.

USES

/etc/passwd To check the correct password, then modifies it to contain the new password.

cut dissects files by fields or columns

FORMAT

```
cut [d=C][od=C] [c=range] [f=range][-nod][-s] [file]
```

DESCRIPTION

cut isolates specified columns or fields of a file, or standard input if no file is specified, and reorders them to standard output.

cut assumes that the input file is divided into columns of characters or fields separated by tabs. You can, however, specify other single character delimiters with the *d=C* option.

cut writes its result to standard output with either the default tab or a delimiter character specified with the *od=C* option.

OPTIONS

d=C C (a single character) specifies input delimiter (default character is TAB, \t). To use a space as a delimiter, enclose the entire option in quotes (see the EXAMPLES).

od=C C (a single character) specifies output delimiter (default character is TAB, \t). To use a space as a delimiter, enclose the entire option in quotes (see the EXAMPLES). Note, delimiters are not used when cutting by columns.

c=range Specifies the character position to extract. Character positions can be specified individually, by ranges or both, e.g., *c=1,2* or *c=1-3,5,7-9*. Character positions are numbered left to right, beginning with 1.

f=range Specifies field number to extract. Fields can be specified individually or in ranges, as with *c=range*. Fields are numbered left to right, beginning with 1.

-nod Suppress output delimiter when cutting by fields. Cutting by columns never has an output delimiter.

cut

-s Suppresses lines without the delimiter. This option is mutually exclusive to character specification.

EXAMPLES

These examples use the text in the following **ttys** file:

```
on:0:/dev/tty0:1/0:z19.alt:: 9600 baud - dianne
on:1:/dev/tty1:1/1:wyse50.80:: 9600 baud - guest
on:2:/dev/tty2:3/0:sw80:: 9600 baud - katz
on:3:/dev/tty3:3/1:sw80:: 9600 baud - ???
off:4:/dev/tty4:3/2:default:: 1200 baud - vacant
on:5:/dev/tty5:3/3:z19.katz:: 9600 baud - ???
printer:6:/dev/tty6:3/4:default::1200 baud - spinwriter
on:7:/dev/tty7:1/2:z19.katz:: 9600 baud - marjie
on:8:/dev/tty8:1/3:z19.alt:: 9600 baud - acronin
```

The following command line uses a colon (:) as the input delimiter and displays the third and first fields, in that order:

```
$cut d=: f=3,1 ttys
/dev/tty0 on
/dev/tty1 on
/dev/tty2 on
/dev/tty3 on
/dev/tty4 off
/dev/tty5 on
/dev/tty6 printer
/dev/tty7 on
/dev/tty8 on
```

cut

This next example uses the colon as the input delimiter, a space as the output delimiter, and displays the first and third fields, in that order:

```
$cut d=: "od= " f=1,3 ttys
on /dev/tty0
on /dev/tty1
on /dev/tty2
on /dev/tty3
off /dev/tty4
on /dev/tty5
printer /dev/tty6
on /dev/tty10
on /dev/tty11
```

DIAGNOSTICS

none

ALSO SEE

paste combines files in columns

NOTES

none

USES

nothing

date

date displays the system date and time

FORMAT

date

DESCRIPTION

date prints the system date and time.

OPTIONS

+%D or **+%d** gives you a numeric date

EXAMPLES

To print the current date and time, type

\$date

Mon Jan 2 1984 13:45:08

DIAGNOSTICS

none

ALSO SEE

setdate sets system date and time

NOTES

date is based on the number of seconds since midnight Jan 01 1980, hence no date before that time can be displayed.

The UNOS **date** command has moved to **/cmd** because the UNIX command of the same name resides in **/bin**.

USES

nothing

debe copies a file or a piece of a file

FORMAT

debe [option=value] [-flag] ...

DESCRIPTION

debe copies the specified input to the specified output performing the requested conversions. The standard input and standard output are used by default.

OPTIONS

-fill Pads every output record with zeroes to the requested size.

-pg Prints a dot each time a record is written to indicate progress.

iskip=# Skips # bytes of the input file before beginning transfer.

oskip=# Skips # bytes of the output file before beginning transfer.

skip=# Combines the functions of *iskip=#* and *oskip=#*.

ivskip=# Skips # bytes at the beginning of each input volume.
The *iskip=#* option still works, but only applies to the first volume, where it adds its value to the value of *ivskip*.

ovskip=# Skips # bytes at the beginning of each output volume.
The *oskip=#* option still works, but only applies to the first volume, where it adds its value to the value of *ovskip*.

ivsize=# Specifies the maximum length, in bytes, of each input volume. When the end of the volume is reached, **debe** asks for the next volume to be mounted. It waits for you to type a "Y" <cr> before it begins the new volume. If you type an "N", it is treated as an end of file condition and **debe** writes any buffered data to output and exits.

debe

ovsize=# Specifies the maximum length, in bytes, of each output volume. At the end of each volume **debe** prints a message requesting that the next volume be mounted. It waits for a "Y" <cr> to be typed before it begins the new volume. If an "N" is typed in response to the message, **debe** stops and the statistics it prints show that more data were read than written.

ibs=# Input block/buffer size in bytes.

obs=# Output block/buffer size in bytes.

bs=# Both in/output block size in bytes.

if=name Input file name.

of=name Output file name.

-noerror Continues transfer despite errors. Error messages will appear on the screen.

count=# Transfers # of input records or until EOF.

-notrunc Does not truncate an already existing output file, enables copying one file into another.

-swab Swaps bytes (except for the last byte in odd block sizes and odd transfers due to EOF).

EXAMPLES

To copy the floppy disk on drive 0 to the cdc virtual floppy, 52 blocks at a time, type: `$debe if=/dev/rcfd0 of=/dev/rcdcv3 bs=52b`

DIAGNOSTICS

Extremely large buffer sizes (*bs=#*) that work in the single volume case can cause the error message:

debe: Program is too large to fit in memory Can't allocate storage

if an *ivsize* is given or if an *ovsize* is specified that is not a

debe

multiple of the buffer size.

When using **debe** with multiple volumes, the following message is printed:

debe: Done with input volume 1. Do you want to continue?

If so, mount volume 2 on /dev/rmf0 before answering.

Answer yes or no: y

Working on input volume 2.

ALSO SEE

nothing

NOTES

When numbers are unspecified, they are assumed to be bytes: *iskip*, *oskip*, *ibs*, *obs*, *bs*, *ivsize*, *ovsize*, *ivskip*, *ovskip*, *skip* and *count*. However, *b* meaning block (512 bytes) and *k* meaning (1024 bytes) can also be used (e.g., *<skip=10b bs=1k*).

USES

nothing

debug debugs a program

FORMAT

debug [*mlist=file*][*-nobj*][*p=pid*][*-unbuf*] *symfile*
[*datafile*]

DESCRIPTION

debug lets you monitor the execution of a program and facilitate interactive debugging. UNOS suspends processing when it discovers a fatal error. You can also suspend a process to debug it and examine a suspended process.

debug takes symbol definitions from the *symfile*, and an optional *datafile* can supply the data. If you do not specify a *datafile*, **debug** uses *symfile* for both. The debugger understands the core file automatically. If you type
\$debug filename

OPTIONS

Command Line Options

The **debug** command takes these options.

mlist=file Reads the file named for macro definitions. The file format has the same requirement for the command **%read**.

-nobj Examines all of the file. By default, **debug** accesses an object file by the address the object file would have when executed.

p=pid Examines the suspended process whose process id is given by **pid**.

-unbuf Opens the data file in the unbuffered mode. The buffered mode is the default mode. Although the buffered mode is more efficient, it prevents you from seeing the changes immediately. Use unbuffered when examining UNOS tables via **/dev/kmem** if you want to note changes as they occur.

debug

DEBUG COMMANDS FORMAT

The **debug** command format consists of an optional expression value followed by an optional command string and ending with a carriage return.

debug works by first reading in the program's symbol table and then prompting you with the special debugger prompt:

DBG>

Expressions

Expressions have values that you construct using the arithmetic operators +, -, *, and the operator @. The operator @ represents the value of the contents of the location. You can represent the value by *[size]* and *[radix]* specification. To avoid unexpected interpretations specify size and radix every time.

Every expression sets the current location. You can reference this location with the . (period) symbol.

The operands of an expression include

- numbers
- the symbol . (period) meaning the current position
- register names in parentheses (valid only if there is a process being debugged)
- names of external variables and functions (valid only if the symbol definitions are present in the symfile given on the command line)

Typical expressions are

```
main
.-20
4(fp)@lx
(pc)
```

Modifiers

You can use the following modifiers with some of the **debug** commands.

unsigned

This modifier specified by the character *u* prints the *[exp]* as an unsigned number. If you do not specify a *u* modifier, values shorter than 4 bytes are sign-extended before printing and the decimal output is signed.

size modifier

This modifier indicates the size of the value. Use the following characters to specify the *[size]* modifier:

c char 1 byte

b byte

w word 2 bytes

s short 2 bytes

i int 4 bytes

l long 4 bytes

The default *[size]* is the last *[size]* specified in the process (initially *i*, int).

radix

This modifier specifies the printed format of the value.

d decimal

o octal

x hex

p pointer - the name of the nearest preceding external symbol and an offset (if not zero)

a ASCII

n do not display, but skip over the proper size item

z prints numbers until a 0 is reached, stops and does not display the last number.

? displays an instruction - ignores *[size]* and advances . (period) by the length of the instruction

The last *[radix]* specified is the default for this field (initially *d*, decimal).

Command Line

The four basic ways of combining *[exp][command string]* are

1. *[exp][command string]*

Sets the value of the current location to the value of *[exp]*, and executes *[command string]*. The location begins at 0. Commands can alter the current location. Type .

debug

- (period) to display the current location.
2. [command string]
Resets the value of the current location to the value it had at the beginning of the last command, and executes [command string].
3. [exp]
Sets the value of the current location to the value of [exp] and repeats the last [command string].
4. <cr>
Leaves the value of the current location unchanged and repeats the last [command string]. For example, if the command is ?, which displays an instruction, each <cr> will display the next instruction.

Command Strings

Multiple commands can appear on a single line and in any order. The only limitation is physical—255 characters.

Some commands accept an optional argument <exp2> before the command. Since the first <exp> on the line is assumed to be the starting position for the command, it is necessary to put a semicolon before <exp2> if it is the first expression on the command line. For example, to display twenty instructions starting at the same address as the last command:

```
DBG> ;20[?]
```

DEBUG COMMANDS

Commands

This section lists and describes commands of the debugger.

@i(:) Separates expressions and/or command strings in the command line.

<=> Prints either the value of [exp] or the current value.

string Prints the quoted string literally.

/ Modifies the location expression or displays the current location.

Built-in Commands

A % character in the command string indicates a built-in

debug

command except for the **!** command that is explained below.

! Passes the rest of the command line **string** to the command interpreter as defined by the environment variable *SHELL* (default is /bin/command) and waits for it to complete.

%breakpoint Sets a breakpoint at the location *[exp]* if specified, or at the current location.

%continue Continues a program for an active process from a breakpoint or suspension.

%define Defines a macro.

%delete Deletes a breakpoint at a specified location expression or at the current location.

%dloc Allows modification or view of memory.

%domain Sets the domain for routine **name** manipulation of local symbols.

%halt Returns to **debug** command level instead of continuing the process.

%kill Ends the current process (child) and returns to the debugger command level.

%list Lists the set breakpoints.

%loc Allows you to modify or view a specified local symbol. When you enter **%loc**, the most current call to the domain routine is found and calculates the locals symbol *[name]* address or register number.

%mlist Lists the specified macros and their definitions.

%newline Starts a new output line.

%pstep Executes similarly to **%step**. The difference between the two commands is in the execution. The **pstep** call is considered a one step command whereas **step** may stop at

debug

each routine until reaching the named routine.

%quit Leaves the debugger.

%read Reads in a file that contains macro definitions.

%registers Prints the current values of the registers for active processes.

%run Executes the program being debugged.

%stack Prints a stack trace for an active process.

%symbol Displays all the symbols in the symbol table. The symbol table shows the symbol name, the symbol type, TXT or BSS or DAT (see *UNOS Developers Manual*) and the symbol address. The symbol "G" terminator indicates a global.

%step Executes a single machine instruction. If an expression precedes the command, the command evaluates the expression then the program steps that number of instructions. A process must be active.

%undefine Removes the definition of the macro *macro_name*.

%write Writes all current macro definitions to a file *filename* in the format required by **%read** or the **mlist=filename** option.

Macros

The debugger provides facilities to create macros. You create macros by combining command strings up to 256 characters. Use the **%define** command or the **%read** command to define these strings. The *mlist=filename* option defines a string when you invoke **debug**. The **%write** command saves all macros used by **debug**.

When the debugger finds a **\$** character in the command string it expects a macro name to follow. Then, the debugger interprets the command string associated with the macro name and executes it.

NOTE: You can nest macros but, nested macros can become recursive. Since there is no way to terminate the recursion, a loop results.

Iteration

The convention for specifying iteration is to bracket a command string and precede it with a count argument. Nest the commands within brackets.

Examples:

```
DBG>;10[/debug0ca/sd%newline]
        Prints 10 instances of a structure
```

```
DBG>;20[?]    Prints the next 20 instructions
```

Environments

If you want to save the current location pointer so you can return to it later, use { } to push and pop the current environment.

{ Pushes the current location pointer and sets the location to the argument provided.

} Pops the location to the saved value and restores the location pointer to the value at the last {, or at the beginning of the command.

; Restores the location but does not pop it.

For example, to print three words and the corresponding instruction:

```
DBG>{/debugso}?
```

Notice how the { pushes the location so that later you can print the values in a different format. You can print unions in a similar manner. If a structure contains a union, the ; command prints it.

```
DBG>/2ld {/ld;/sd;/10ca} .+10/sd
```

Since the ; command resets the pointer to the beginning of the union, you must use .+10 to advance to the next element of

debug

the structure. Step through the structure explicitly, because the { } combination has no effect on the location pointer.

```
DBG>/21d {/1d;/sd}/10ca /sd
```

OR

```
DBG>/21d {/1d;/sd;/10ca}/10cn /sd
```

EXAMPLES

```
$debug hello
```

```
DBG>!cat hello.c
```

```
main ()
```

```
{  
    printf ("Hello, world.\n");  
}
```

```
DBG>main/5?
```

main:		
118	LINK	FP,#0xFFFFC
122	TST.B	-256(SP)
126	MOVE.L	#0x40001C,(SP)
132	JSR	printf
138	UNLK	FP

```
DBG>printf%br
```

```
DBG>%run
```

```
Debugging process 662.
```

```
Process stopped at printf+4.
```

```
bpt/trace.
```

```
DBG>%sta
```

```
Process stopped at printf+4.
```

```
bpt/trace.
```

```
printf(0x400004)
```

```
main(0x1,0xDFFF6A,0xDFFFE0)
```

```
DBG>0x400004/zca
```

debug

Hello, world.\012

DBG>%kill

Process 662 terminated.

DBG>%quit

\$

DIAGNOSTICS

none

ALSO SEE

asm translates assembly language files

ccom68 compiles C programs

gcc compiles C programs using the Green Hills C compiler

wcc compiles C programs using the Whitesmiths C compiler

UNOS System DEBUG Manual

NOTES

none

USES

nothing

delete

delete deletes files

FORMAT

delete [options] file1...filen

DESCRIPTION

delete removes file names from their parent directories. If you are deleting the only name a file has, the file pointed to by that file name is removed from the filesystem.

Unless the *-recursive* flag is given, files which contain directories will only be removed if the directory is empty.

If no files are specified on the command line, the user is prompted for file names on standard input. The > prompt, however, will only be issued if the *-interactive* option is given.

You must have write permission to the directory that contains the file you are deleting (parent directory). You must also have write permission to the file you are removing. Although the *-force* option (described below) lets you remove a file for which you don't have write permission, it won't help you remove a file if it's in a directory for which you don't have write permission.

OPTIONS

All options can be abbreviated to their first letter.

-force Use this option to remove files for which you do not have write permission.

-help Prints, to standard error, a few lines listing all the command options **delete** accepts.

-interactive Prompts the user for more files after removing any files given on the command line. This option implies *-query*.

delete

- query* Asks whether a file should be removed. A response of either yes or y removes the file; any other response is interpreted as negative. -
- recursive* Removes non-empty directories by removing the entire subtree defined by that non-empty directory.
- silent* Prints only error messages. Use this option if you don't want exit messages displayed.
- verbose* Lists files that have been removed.

EXAMPLES

To remove the file test.j, type
`$delete test.j`

When you use the following command line, **delete** continues to prompt for file names until you type a **^D** or a blank line.
`$delete -i >x.j >y.j > ^D]`

The following command line will also work:
`$delete < filename`

DIAGNOSTICS

none

ALSO SEE

nothing

NOTES

You can use **delete** to delete eventcount files and special device files.

USES

nothing

describe

describe displays manual page for a command or subroutine

FORMAT

`describe [-c] [-help] [-l] [-s] [pattern]`

DESCRIPTION

describe provides on-line documentation for commands and subroutines. To review the documentation, type **describe**, an optional argument and the name of the command or subroutine.

The **-c**, **-l** and **-s** options locate a documentation file by searching the docref in the directory associated with that argument. By default **describe** searches all three directories in a specified order. The search stops at the end of the directory where **describe** finds a docref. If you specify any or all of the options, **describe** searches only those options specified in the default order.

OPTIONS

-l Indicates the pattern is in your present directory.

-s Indicates that the pattern is the name of a subroutine.

-c Indicates that the pattern is the name of a command.

pattern Indicates the subroutine or command for which you want on-line documentation.

EXAMPLES

To display the help file for the **copy** command in your current directory, type

```
$describe -c copy
```

To display the help file for the **format** command instead of the **format** subroutine type

```
$describe -c format
```

describe

To display the help file for the **format** subroutine, type
\$describe -s creat

DIAGNOSTICS

none

ALSO SEE

User's Guide to UNOS

NOTES

In a no argument format, **describe** seeks a documentation file

- in the current directory
- then in the commands directory (default **/doc/cmds**)
- finally in the subroutine directory (default **/doc/subs**)

Once **describe** finds a documentation file match it stops searching. Place pattern matching in quotes, for example:
\$ describe "f*"

describe is unable to search multiple files.

If you do not specify **-s** or **-c**, **describe** checks both command and subroutine references, which results in displaying on-line documentation for both when they have the same name (e.g., **format**).

describe without a pattern works the same as the command **describe describe**.

USES

docref In the current directory if one exists.

/doc/cmds/docref For system commands.

/doc/subs/docref For system subroutines.

difference

difference indicates how two text files differ

FORMAT

difference file1 file2

DESCRIPTION

difference shows what lines differ between two files.

OPTIONS

none

EXAMPLES

This example shows using **difference** to compare the contents of two files, *file1* and *file2*:

file1	aa	file2	bb
	bb		cc
	cc		dd

```
$difference file1 file2
----- 3 lines changed at 0-2 from:
aa
bb
cc
----- to:
bb
cc
dd
```

DIAGNOSTICS

none

ALSO SEE

nothing

difference

NOTES

No output is generated if the files are the same.

USES

nothing

diskusage

diskusage counts blocks used by files

FORMAT

diskusage [directory1...directoryn]

DESCRIPTION

diskusage shows the number of blocks the named or current directory is using.

It lists the names and blocks used (including any subdirectories), and on the final line, the total usage.

OPTIONS

none

EXAMPLES

```
$diskusage /etc
 11 /etc/maillists
  7 /etc/mailboxes
 14 /etc/ttydesc
238 /etc
Total blocks: 238
```

DIAGNOSTICS

diskusage: cannot open <filename>

ALSO SEE

space displays amount of free space in the file system

NOTES

When filenames are given to **diskusage**, they will be counted and added to a total, but not otherwise displayed.

diskusage

USES

nothing

download

download loads and starts the IOP code

FORMAT

download [*board*=*n*] [*iop*=*pathname*] [*tty*=/*dev/name*]

DESCRIPTION

download loads the specified file into iop number *n* and starts the mbx driver running. The specified *tty* must be one of the devices associated with the board. **download** is a system utility for superuser only.

OPTIONS

board=*n* Specifies the VME IOP board number to download. Currently accepts numbers zero through 7. Default is zero (0).

iop=*pathname* Specifies the path of the file to download. Default is /sys/iop.VME

tty=/*dev/name* Specifies a port name on the board for initialization. Default is /dev/tty4.

EXAMPLES

```
#download board=1 iop=/sys/iopfile tty=/dev/tty8
```

DIAGNOSTICS

Vectors refused.

Absolute access to vector memory on the main MPU has failed.

I/O refused.

Absolute access to I/O space on the main MPU has failed.

Wake IOP #*n* timed out!

IOP #*n* does not respond at all.

download

Can't open IOP program file: 'pathname'.

The iop code file 'pathname' does not exist or is protected.

Load IOP #n timed out!

IOP #n will not finish the write memory command or respond to the download command.

ALSO SEE

nothing

NOTES

none

USES

nothing

dump

dump dumps selected parts of an object file

FORMAT

dump [-acfg hlors t] [-z *name*] files

DESCRIPTION

dump dumps sections of object files or archived object files. The sections dumped are specified through the options outlined below.

OPTIONS

- a Dumps the archive header of each member of each archive file argument.
- c Dumps the string table.
- f Dumps each file header.
- g Dumps the global symbols in the archive symbol table.
- h Dumps section headers.
- l Dumps line number information.
- o Dumps each optional header.
- r Dumps relocation information.
- s Dumps section contents.
- t Dumps symbol table entries.
- z *name* Dumps line number entries for the function *name*.

The following modify the options listed previously.

- d *number* Dumps the sections starting at *number* and ending with either the section specified with +*d* modifier, if +*d* is given, or with the last section number.

dump

- +d number** If present without the **-d** modifier, dump from the first section to the section specified with **+d**. If **-d** is present, dump from the section specified with that option to the section specified by the **+d** option.
- n name** Dumps only information pertaining to *name*. This modifier can be used with the **-h**, **-l**, **-r**, **-s**, and **-t** options.
- t index** Dumps the indexed symbol table entry only, starting at *index*.
- +t index** Specifies the ending index for the **-t** modifier. If **-t** is absent, **+t** starts from the first entry.
- u** Underlines file names for emphasis.
- v** Dumps information using symbolic rather than numeric representation. This modifier can be used with all dumping options save **-s** and **-o**.
- z name, number** For function *name* dump the line number entry *number* or the range of entries starting at *number*.
- +z number** Dumps line numbers starting at either the function *name* or the function *number* specified by **-z**, up to *number* specified by **+z**.

You can use an optional whitespace to separate an option from its modifier. and to separate the *name* and *number* fields of the **-z** option.

EXAMPLES

An example of **dump** output format follows:

```
shell> dump -t aprog
```

*** SYMBOL TABLE ***								
Indx	m1	Name	Value	Scnum	Type	Sclass	Numaux	Name
	m2	Name	Value	Scnum	Type	Sclass	Numaux	Name
	a	Tagndx	Misc	Misc	Fcnar	Fcnar		

dump

aprogram:

0 ml	.file	2	-2	0	103	1
2 ml	.file	599	-2	0	103	1
4 ml	start	0	1	0	3	0
5 ml	.file	599	-2	0	103	1

etc.

DIAGNOSTICS

self-explanatory

ALSO SEE

nothing

NOTES

none

USES

nothing

echo displays arguments

FORMAT

echo [-nnl] message

DESCRIPTION

echo displays its arguments.

It can be used to echo its arguments onto the terminal screen, or, when inserted in a command file, to reflect the progress of the program.

OPTIONS

-nnl suppresses printing of newline

EXAMPLES

To name all the files that match a pattern, type

```
$echo d*
```

```
date debug delete diskusage dump
```

To display comments on the terminal screen, type

```
$echo "When in the course of human events..."
```

```
When in the course of human events...
```

The following command file can be used to alert the user of the progress of a command file:

```
format file.a > file.format
```

```
echo "file.a has been formatted"
```

```
move file.format paper
```

```
echo "file.format has been moved to paper"
```

DIAGNOSTICS

none

echo

ALSO SEE

cat concatenates files

NOTES

none

USES

nothing

email sends and receives electronic mail

FORMAT

```
email [b=mailboxname] [-checknewmail][-newmail]
email names
email file=messagefile [-delete] names
email send=randomfile [-delete] [names]
email auto=randomfile [-delete] names
```

DESCRIPTION

email lets you send and receive messages.

OPTIONS

Typing the command followed by *b=mailboxname* lets you specify the mailbox (rather than the default mailbox, called *active*). If you use the *-checknewmail* option, there is a check for new messages. If there are new messages, you are placed in the **email** program, otherwise you remain at command level. The *-newmail* option is the same as *-checkmail*, only you remain in the **email** program.

Typing the command followed by *name(s)* lets you compose and send messages to the listed users. You are automatically placed in the editor with the **To:** field filled in. The Editor is exited as soon as the message is sent.

The *file=* option lets you send a file that is already in the form of a message (it begins with a **To:** line). Note that the *send=* and *auto=* options are preferred since they let you send messages in any format.

The *send=* option lets you specify a file to be sent and optionally deleted. *send=* will send any file. If you do not specify recipients, you are placed in the editor with the **To:** field to be filled in.

The *auto=* option automatically sends the file you specify to the listed users. A subject line will not be added to the message and

email

no formatting will be done (as is when the *send=* option is used). You only need to have read access to send a file with the *auto=* option.

To send a certified message (you will automatically be notified when the recipient has read it), precede the name in the **To:** field with a percent sign (%).

The **email** program offers you a rich set of tools for handling messages. Consult the *User's Guide to UNOS* for a description on how to use the program.

EMAILER COMMANDS

Number (#) can be a single number, 1-9.

Ranges of numbers can be any list of comma separated numbers, "3,5,7". Dashes can be used to specify, from the first to the number, "-4" or from the number to the last, "4-" or the numbers in between, "4-7". An asterisk (*) or a dash (-) means all entries.

Name may be the name of an alternative mailbox or the name of a file. A file is created if one did not already exist.

Command can be any UNOS command which is then executed through the Command Interpreter.

Special characters can be used

"." means current message

">" means next

"<" means previous

"^" means first

"\$" means last

Confirmation can be any of the following:

Y

y

TAB

space

<cr>

linefeed

email

All commands are one character long; if you type more than one character, the succeeding characters are applied to the command's next prompt (if there is one).

a # Answer Message.

Enables you to answer a specified message. The Email displays a template with the To: field containing the original sender's name, cc: field user's name, and the Subject field, Re: preceding the original subject.

b name Change Current mailbox to:

Lets you choose to either change to another mailbox; the Emailer creates one if none existed, or list mailboxes. Before changing to the new mailbox, all deletes become final.

c # Change Message.

The Emailer displays the specified message on the screen for you to edit.

d range Delete Message.

The Emailer marks the specified message or range of messages as deleted, that is, the letter 'D' precedes the entry. When you exit the Emailer or change to another mailbox, these messages are deleted.

e name Edit Distribution List.

Allows you to edit the distribution list, either your personal (local) or the system's (Global).

f range Forward Message.

The Emailer puts you into the editor with a copy of the specified message(s). The To: and subject: fields are empty.

g # Goto Message.

The Emailer makes the specified message the current message (an asterisk marks the current message).

h range printer Hardcopy Message.

The Emailer prints the message(s). The default printer is used, unless you specify the optional printer.

email

l range List Message in Current Mailbox.

The Emler displays the headers in the range of the current mailbox. To see all the headers, type a space.

m range name Move Message to Mailbox <name>.

The Emler moves the specified messages to the named mailbox and deletes them from the current mailbox.

n New Mail.

The Emler reads in the new messages and displays the header.

p range name Puts Message into file <name>.

The Emler puts the specified messages into the named file. It creates the file if it did not exist.

q Quit (requires confirmation).

Exits the Emler and returns to command level.

s Send Message.

The Emler puts you into the editor and gives you a template with the To: and subject: fields empty.

t range Type Message.

The Emler displays the messages on the screen. To stop the output type the letter s.

u range Undelete Message.

The Emler recinds all deletions.

v # Enter Editor With Message.

Allows you to view a message, but not alter it. You cannot specify ranges.

x <command> Execute a command.

Allows you to execute a command from within the Emler.

? Displays the help file.

EXAMPLES

To send a certified message:

```
To: %dan
cc: katz
Subject: Important message
```

When the message is read, you'll receive the following message:

```
Time: Wed Oct 31 1984 15:36:39
From: dan
To: katz
Subject: Certified: Important Message
```

Your message has been read.

To see a list of the headers in the current mailbox, type l, wait for the prompt and then press the space bar.

```
Email> List message #s (space=): * ...
See displayed:
```

```
active box
  1 - Tue Feb 2 - jeff      -updated fscheck
  2 - Thu Feb 4 - allen    -Subroutines
 *3 - Thu Mar 30 - morrie  -new users on V
```

DIAGNOSTICS**Message not sent**

When there are no recipients of a send or answer.

<#> does not exist!

When you specify a non-existent message number.

Bad range string <#-#>

When an illegal range is specified.

<command> -not a command!

When an illegal command is requested.

email

ERROR: PROBLEM SENDING TO USER OR DISTRIBUTION LIST

<name>

When **name** is not a user in **/etc/passwd** or a valid distribution list (local or global).

ALSO SEE

write sends a message to specified users

wall sends a message to all users currently logged on
User's Guide to UNOS

NOTES

If you sent a mail message to a logged on user, **email** will send a message to that terminal, informing him or her of the new mail.

The editor Email uses is specified by the EDIT environment variable. If there is no EDIT variable, Email uses TED.

If you have UniverseNet, you can use the **email** command over the network. See the *UniverseNet Network Administrator's Guide* for more information.

USES

/tmp/logins To know who is logged on.

/doc/cmds/email.help The **email** help file.

/etc/passwd To identify sender and locate recipient.

/etc/maillists Used for global distribution lists.

/etc/mailboxes Used for new mail.

The following files are in the user's home directory

.mail.* Used for mailboxes.

.maillist.* Used for local distribution lists.

email

Also the following commands are used

/bin/ted To edit messages.

/bin/lpr To enqueue files for printing.

false

false returns an error indication .

FORMAT

false

DESCRIPTION

false does nothing but return an exit status of 1. It is primarily useful in conditional expressions in shell scripts.

OPTIONS

none

EXAMPLES

See the UNIX **sh** command.

DIAGNOSTICS

none

ALSO SEE

true returns a success indication

sh UNIX Bourne shell

csh UCB C shell

NOTES

none

USES

nothing

find

find locates files

FORMAT

find [directory_list] [function]

DESCRIPTION

find applies the boolean FUNCTION to each accessible file within the file system subtrees whose roots the DIRECTORY_LIST specifies. By default, **find** displays the full pathname of any file that causes the value of FUNCTION to be TRUE.

DIRECTORY_LIST is a list of directory pathnames. Each directory in the list specifies the root of a file system subtree. Assume the current directory if DIRECTORY_LIST is unspecified.

FUNCTION is a boolean function constructed from boolean terms and operators according to the rules of boolean algebra. Command line options specify the boolean terms and operators that comprise FUNCTION. Assume a TRUE value for all files with an unspecified FUNCTION.

find supports two types of boolean terms, attribute specific terms and execution specific terms.

An attribute specific term establishes a condition pertinent to a specific attribute of a UNOS file (size, type, modes, etc.). The value of the term depends on the attributes of the file to which FUNCTION is currently being applied. The value of the term is TRUE if the pertinent attribute of the file satisfies the condition established by the term. The value of the term is FALSE if the file attribute fails to satisfy the condition established by the term.

An execution-specific term causes the execution of a UNOS command or program. The file that applies to FUNCTION may execute or pass its file name to another program as a command line argument. The value of the term is a boolean function of the exit status of the executed program. The value of the term

find

is TRUE if the program's execution results in a value of zero as its exit status. The value of the term is FALSE if the program's execution results in a non-zero value exit status.

find supports several boolean operators, specifically, not, and, or, and parentheses (precedence ordering). These operators combine with boolean terms to form an infix boolean expression which defines FUNCTION. You evaluate the expression left to right, according to precedence.

OPTIONS

Attribute Specific Term Options

Attribute Specific Term Options are command line arguments that specify an attribute specific boolean term.

links=[[/-]n] If the argument is unsigned, this term has a value of TRUE if the file has exactly 'n' links. Otherwise, the value of the term is FALSE.

If the argument is prefixed by a '+' sign, this term has a value of TRUE if the number of links to the file is greater than 'n'. Otherwise, the value of the term is FALSE.

If the argument is prefixed by a '-' sign and the number of links to the file is less than 'n', this term has a value of TRUE. Otherwise, the value of the term is FALSE.

size=[[/-]n] If the argument is unsigned and the file is using exactly 'n' blocks (512 bytes/block), this term has a value of TRUE. Otherwise, the value of the term is FALSE.

If the argument is prefixed by a '+' sign and the number of blocks used by the file is greater than 'n', this term has a value of TRUE. Otherwise, the value of the term is FALSE.

If the argument is prefixed by a '-' sign and the number of blocks used by the file is less than 'n', this term has a value of TRUE. Otherwise, the value of the term is FALSE.

find

If you omit the argument, the default value of zero is used.

time=[*[/-]**n*] If the argument is unsigned and the file was last modified exactly '*n*' days ago, this term has a value of TRUE. Otherwise, the value of the term is FALSE.

If the argument is prefixed by a '+' sign and if the date of the last modification of the file is greater than '*n*' days ago, this term has a value of TRUE. Otherwise, the value of the term is FALSE.

If the argument is prefixed by a '-' sign and if the date of the last modification of the file is less than '*n*' days ago, this term has a value of TRUE. Otherwise, the value of the term is FALSE.

If you omit the argument, the default value of zero is used.

type=*n* The value of the term is TRUE if the type of the file is one of these single character codes:

b	buffered device file
c	unbuffered (character) device file
d	directory file
e	eventcount file
f	plain file
p	named pipe

If the type of the file is not one of these single character codes then the value of the term is FALSE.

You must include this argument.

name=*filename* The value of the term is TRUE if the name of the file matches the argument *filename*. Otherwise, the value of the term is false. You must place UNOS pattern matching characters between quotes to prevent the Command Interpreter from expanding them.

You must include this argument.

find

user=[uname] The value of the term is TRUE if the owner of the file is specified by the *uname* argument. You can specify a name or a number. Otherwise, the value of the term is FALSE.

If you omit the argument, the value of the term is TRUE for all files.

group=[gname] The value of the term is TRUE if the group owner of the file is specified by the *gname* argument. You can specify a name or a number. Otherwise, the value of the term is FALSE.

If you omit the argument, the value of the term is TRUE for all files.

perm=[[+/-]r[[+/-]w[[+/-]x],
[[+/-]r[[+/-]w[[+/-]x],
[[+/-]r[[+/-]w[[+/-]x]

The value of the term is TRUE if the access modes of the file match the access modes specified by the argument. Otherwise, the value of the term is FALSE.

If you omit the argument, the value of the term is TRUE for all files.

Execution Specific Term Options

Execution Specific Term Options are command line arguments that specify an execution specific boolean term.

exec=command_line The value of the term is TRUE if the execution of the argument 'command_line' results in a zero value as an exit status.

Otherwise, the value of the term is FALSE.

If the string *\$1* appears in *command_line*, **find** substitutes the file name.

You may specify the option alternately in the form *-exec command_line*.

ok=command_line This option is similar to the *exec* option described above, however *find* displays:

"'command_line'?"

and executes *'command_line'* only if the user responds by typing *"y<cr>"*.

The value of the term is **TRUE** if the execution of the argument *command_line* results in a zero value as an exit status. Otherwise, the value of the term is **FALSE**.

If the string *\$1* appears in *'command_line'*, **find** substitutes the name of the file.

You may specify the option in the form
"-ok command_line".

-print The value of this term is always **TRUE**. The full pathname of the file prints, followed by a newline.

-printnnl The value of this term is always **TRUE**. The full pathname of the file prints, followed by a space. Because it does not print a newline, this term is useful if you want to pipe output to a command script that requires a single input line.

Boolean Operator Options

Boolean Operator Options are command line arguments that specify a boolean operator. The operators are listed in order of precedence.

Boolean expressions within parentheses are evaluated before any other evaluation occurs. Parentheses must be within quotes.

(expression) Precedence grouping

-! Prefix NOT.

Compliments a term. For example, the value of *size=10* is **TRUE** for all files using 10 blocks. The value of *-! size=10* is **FALSE** for all files using 10 blocks and **TRUE** for all other files.

find

-a Infix AND operator.

The first term in the expression is evaluated. The next term in the expression is evaluated only if the value of the first term is TRUE. The result of the operation is TRUE only if the values of both terms involved in the operation are TRUE.

The result of the operation is FALSE if the values of either or both of the terms involved in the operation are FALSE.

-o Infix OR operator.

The first term in the expression is evaluated. The next term in the expression is evaluated regardless of the value of the first term. The result of the operation is TRUE only if the value of either term in the operation is TRUE. The result of the operation is FALSE only if the values of both of the terms in the operation are FALSE.

If an operator is omitted between two terms in an expression, assume the infix AND operation.

It is only necessary to prefix the first Boolean Operator Option in an expression with a dash (hyphen). The dash is optional for all remaining Boolean Operator Options in the expression.

EXAMPLES

Displays the full pathnames of all files with the name "foo" contained in the file system subtree whose root is /usr.

```
$ find /usr name=foo
```

Displays the full pathname of all files with exactly 2 links contained in the file system subtree whose root is the current directory.

```
$ find links=2
```

Displays the full pathname of all files that use more than 5 blocks and that have not been modified within the past year (365 days) contained in the file system subtree whose root is the directory /usr/ed.

```
$ find /usr/ed size=+5 -a time=+365
```

find

Displays a long listing of all eventcount files contained in both of the file system subtrees whose roots are the directories `/usr/dano` and `/usr/mcr`. Notice that quotes surround the `"exec"` option to prevent the shell from interpreting the embedded white space in the list command line.

```
$ find /usr/dano /usr/mcr type=e -a "exec=list -l $1"
```

Deletes any file owned by group `"sys"` with a name beginning with the characters `"tmp"`, or any UNOS object file, contained in any file systems currently mounted.

```
$find / "(" group=sys -a name=tmp* o name=*.j ")" -exec  
"delete $1"
```

Put the parentheses in quotes to prevent their interpretation by the shell. The parentheses are necessary as the infix OR operator has a lower precedence than the infix AND operator implied between the `name=*.j` term and the `-exec` term. Without parentheses to group the expression, only the object files would be deleted and leaving those files owned by group `"sys"` with names beginning with `"tmp"` unaffected. Besides the implied infix AND operator previously mentioned, notice that

- both `"name"` terms utilize the standard UNOS pattern matching metacharacter `"*"`
- the infix OR operator was specified without a dash
- an alternate specification of the `"exec"` term was used.

DIAGNOSTICS

Diagnostic messages are many and varied, but self-explanatory.

find: Bad pattern in 'name='.

The argument to the `"name"` option was omitted.

find: Bad type '<type>' in 'type=<type>'.

You omitted the argument to the `"type"` option or you incorrectly specified the argument.

**find: Non numeric character '<char>' in
'<option>=<char>'.**

You used a non-numeric character other than `'+'` or `'-'` in an argument to the `'time'`, `'size'`, or `'links'` option.

find

find: User '<string>' not in /etc/passwd in
'user=<string>'.

No argument to the user option was found in the /etc/passwd
file.

find: Group '<string>' not in /etc/group in
'group=<string>'.

No argument to the group option was found in the /etc/group
file.

find: Bad mode character '<char>' in
'perm=<modestring>'.

The '<modestring>' list of access modes supplied as an argument to
the perm option contains the invalid mode character '<char>'.

ALSO SEE

See *User's Guide to UNOS* for further information on the UNOS
file system structure and pattern matching.

NOTES

find does not report access violations.

Being unable to open a file is not a fatal error. A fatal error
would destroy the utility of the program.

USES

/etc/passwd To find user names.

/etc/group To find group names.

fmtdisk

fmtdisk formats a hard or floppy disk

FORMAT

fmtdisk device [interleave] [-force] [pvh=filename]

DESCRIPTION

WARNING!! THIS COMMAND ERASES ALL DATA ON THE ENTIRE DEVICE SPECIFIED.

fmtdisk causes the disk controller to reformat all of the specified disk. This involves writing the sector ID fields, address marks, and data fields.

The default interleave results in the best performance for each drive type. Refer to the appropriate disk controller manual for additional interleaves. The disk controller will use the numeric interleave you specify to control the numbering of the sectors within each track. You can vary the number to optimize a particular application.

Because formatting destroys all existing data, **fmtdisk** always asks the question,

Format <device>: are your sure?

fmtdisk proceeds only if you type the word yes. Any other response results in termination of the command, leaving the device untouched.

The CC-1 driver may need more information before it can correctly format Winchester disks. The disk normally contains a PVH file (physical volume header) that describes attributes of the particular disk drive. **fmtdisk** cannot just read in the pvh so, always use the pvh=option when formatting a Winchester disk. The PVH file gives the driver a complete description of the disk.

The sys/pvh provides pvh files for the standard disk types. Disk types include cb, cc, cd, cj, ck and cq. The 60 and 120 megabyte

fmttdisk

disks (cj and ck) perform best when you format the disk using a special version of the PVH files (cj.61sec and ck.61sec). The special version PVH files arrange the disk in a way that saves a rotation on every head-switch.

For a description of the format of the PVH files. see the command page for **write__pvh**.

OPTIONS

-force suppresses the message **Format <device>: are you sure?** when formatting a disk.

pvh=filename allows you to name a pvh file.

EXAMPLES

To initialize a double-sided double-density floppy, first format the floppy:

```
$fmttdisk /dev/rcfd0
```

```
Format /dev/rcfd0: are you sure? yes
```

Then create a new file system structure on the floppy:

```
$makefs /dev/cfd0 1989
```

To format a DK-120 (120 megabyte Fujitsu drive):

```
# fmttdisk /dev/rck0 pvh=/sys/pvh/ck.61sec
```

```
Format '/dev/rck0': are you sure? yes
```

Then you will need to write the bad blocks, the PVH files and the partitionings:

```
# /sys/write_pvh /sys/pvh/ck /dev/rck0
```

```
# /sys/write_bad_blocks saved_bad_blocks /dev/rck0
```

```
# /sys/write_partition type=fujitsu_130 /dev/rck0
```

fmtdisk

DIAGNOSTICS

If the disk controller encounters an error, the driver may produce error messages on the system console (tty0), and **fmtdisk** prints **fmtdisk: An I/O error occurred. Format of <device> failed.**

ALSO SEE

Device Organization chapter in the *UNOS System Concepts Manual*.

NOTES

Formatting destroys all information stored on the entire disk, not just the specified portion. Thus formatting **/dev/cb0** also formats (and destroys all data on) **/dev/cb1** and **/dev/cb2**. The bad block list (if any) will also be destroyed and should be recreated. **fmtdisk** also destroys the bad block disk, so you must recreate it. Only the system administrator can format a hard disk.

USES

nothing

format

format formats text files for printing

FORMAT

format [*dev=name*] [*from=#*] [*to=#*][*-f* or *-F*] [*file1..n*]

DESCRIPTION

format reads the files listed in the command line (or standard input) and formats text on standard output.

Command Line Options

The argument *dev=name* specifies the output device. The following are recognized devices:

cent Centronics 704 printer, all special fonts print as normal characters.

spin NEC Spinwriter 7715 printer, prints all four special fonts.

heath the *H19* or *Z19* CRT terminal, all special fonts appear as reverse video.

The argument *from=#* specifies the starting page number and the argument *to=#* specifies the ending page number.

The argument *-f* or *-F* puts a form feed at the end of the page.

Because command line options must precede the file or files to which they apply, different characteristics can be specified for each file.

format

Input Commands

Lines of the input file are interpreted as:

1. A macro definition begins with #
2. A user defined macro begins with \$
3. A formatter command begins with a .
4. A variable value begins with @%
5. A blank line begins with a blank line in the output
6. A line beginning with white space begins a new line in the output
7. Regular text begins with a word.

Macros

Macros are user defined.

The user can group any valid **format** commands in brackets and call the contents by a unique name. Macros are defined by the following format:

```
#macro_name[format command  
format command  
format command]
```

The **macro_name** can be any name up to 16 characters in length. The macro name can also include arguments, as in the following:

```
#macro-name(title)[
```

These arguments represent expressions in the body of the macro.

Macros can be defined and placed in the **format** input file before the text or alternatively, they can be placed in a separate file which is then included in the input file by using either the built-in

```
$include("filename")
```

macro at the beginning of the input file (see **EXAMPLES**) or the diversion command **.ri** (see Diversions).

Variables

A variable is a number that can be used as a value in an expression or its value can be inserted in the body of the text. The formatter recognizes user defined variables. To use the variable as an expression simply use its name after a command, and to insert its value in the text precede its name with "%a%" at the left margin.

Format Commands

Command lines take the form:

.xx <argument>

where **.xx** is a two character name and the optional number, string, variable, or filename, (depending on the command) is placed at the left margin.

Line/Page Length

.ll [+/-]# Line length

Sets the line length. The argument specifies the number of columns per line. An unsigned argument must be non-zero and specify an absolute number of columns per line. If prefixed by a '+' or by a '-', the argument specifies a number of columns per line relative to the current line length. The argument may not be omitted. The default line length is 75 columns.

.pl [+/-]# Page length

Sets the page length to the argument. The argument specifies the number of lines per page. An unsigned argument specifies an absolute number of lines per page. If prefixed by a '+' or by a '-', the argument specifies a number of lines per page relative to the current page length. The argument may not be omitted. The default page length is 66 lines.

Margins/Indentation

.lm [[+/-]#] Sets the left margin

The argument specifies the column position of the left margin. An unsigned argument specifies an absolute column number. An argument prefixed by a '+' or a '-' specifies a column position relative to the current left margin. Omitting

format

the argument or an absolute specification of zero, sets the left margin position to column 1.

.in **[[=/-]#]** Sets the column indentation.

The argument specifies the column indentation position. An unsigned argument specifies an absolute column number. An argument prefixed by a '+' or a '-' specifies a column position relative to the current column indentation. Omitting the argument sets the column indentation to the left margin position.

.ti **[[=/-]#]** Sets the temporary column indent

The argument specifies a temporary column indentation position that applies only to the next output text line. An unsigned argument specifies an absolute column number. An argument prefixed by a '+' or a '-' specifies a column position relative to the current column indentation. Omitting the argument sets the temporary column indentation to the left margin position.

.ta **n,m..** Multiple tabs

Sets tab stops for use with the tab key. Any number of tab stops can be specified. To regain the default setting (every 8 spaces), tabs must be set at 8,16,24,32,40,48,56,64,72,80...}

Spaces

.[sp]# Space

Leaves argument number of blank lines in the text. Omitting the argument leaves one blank line.

.ms **[[=/-[#]** Sets multiple line spacing

The argument specifies the number of carriage returns that appear after each output text line. An unsigned argument specifies an absolute number of carriage returns. An argument prefixed by a '+' or a '-' specifies a number of carriage returns relative to the current multiple space specification.

.pi **#** Picture Leaves argument number of blank lines in the text. If there is not enough room on this page, put it out as is, and begin a new page, leaving argument number of

blank lines at the beginning of the new page.

Fill/No Fill

.fi Fill

Causes line filling to be performed. Text from following lines will be added to the current line until a full line is created. Note that this is independent of the form of adjustment, so it is possible to get filled, centered lines as well as non-filled but justified lines (default mode is fill).

.nf No fill

Turns off filling.

Adjust/No Adjust

.ad Adjust

Causes the text to be justified on both margins. This is mutually exclusive of the commands **na**, **rj**, and **ce**, (default mode is adjust).

.na No adjust

Causes no attempt to be made at justifying either margin. This is mutually exclusive of **ad**, **rj**, and **ce**.

.rj Right justify

Causes the text to be justified to the right margin. This is mutually exclusive of **na**, **ad**, and **ce**.

.ce Center

Causes the text to be centered between the margins. This is mutually exclusive of **na**, **rj**, and **ad**.

Change Bars

.cb Change bar

Causes the subsequent text lines to be preceded by a vertical bar.

.nc No change bar

Turns off the change bar function.

format

Breaks/Need

.bp Break page

Causes the current page to be output without additional text (except for footers) and a new page begun.

.br Break line

Causes the current line to be output without additional text (except for footers) and a new line begun.

.ne # Need

If there is not the specified number of lines left on the page, put out the current page and begin a new page. A space command may be ignored when it follows a need command.

Character Translation

.tr **ab** Translate character

Translates character a to character b on output only. (Do not leave any whitespace between the two characters.) For example, this enables you to handle initial periods by using a non-significant character such as . in the source text. In that case, **.tr ~.** would translate all source file ~ into periods on output.

.et End translate

Turns off the translation.

Headers

The format of a header/footer specification is 'l'c'r' where 'l' is any character not appearing in the header/footer and l c r are left justified, centered, right justified parts of the header. Inside the specification \p is expanded as the current page number, \d is expanded as the current date, and \t is expanded as the date plus the current time.

.he 'l'c'r' Header

Sets the header for both even and odd pages to be the argument.

format

.eh 'l'c'r' Even header

Sets the header for the even pages to be the argument.

.oh 'l'c'r' Odd header Sets the header for the odd pages to be the argument.

.fh Force header

Forces the most recently defined header to be put out at the top of the current page. New headers otherwise take effect only when a new page is begun.

.m1 [+/-]#

Sets the number of blank lines which appear before the page header. The argument specifies the number of blank lines. An unsigned argument must be non-zero and specifies an absolute number of blank lines. An argument prefixed by a '+' or a '-' specifies a number of blank lines relative to the current spacing value. The argument cannot be omitted. Three blank lines appear before the header by default.

.m2 [[+/-]#]

Sets the number of blank lines which appear after the number of blank lines that appear after the page header. The argument specifies the number of blank lines. An unsigned argument specifies an absolute number of blank lines. An argument prefixed by a '+' or a '-' specifies a number of blank lines relative to the current spacing value. If the argument is omitted no blank lines appear after the header. Two blank lines appear after the header by default.

Footers

.fo 'l'c'r' Footer

Sets the footer for both even and odd pages to be the argument.

.ef 'l'c'r' Even footer

Sets the footer for the even pages to be the argument.

.of 'l'c'r' Odd footer

Sets the footer for the odd pages to be the argument.

format

.m3 [=/-]#

Sets the number of blank lines that appear before the page footer. The argument specifies the number of blank lines. An unsigned argument specifies an absolute number of blank lines and must be non-zero. An argument prefixed by a '+' or a '-' specifies a number of blank lines relative to the current spacing value. The argument cannot be omitted. Three blank lines appear before the footer by default.

.m4 [[+/-]#]

Sets the number of blank lines that appear before the page footer. The argument specifies the number of blank lines. An unsigned argument specifies an absolute number of blank lines. An argument prefixed by a '+' or a '-' specifies a number of blank lines relative to the current spacing value. If the argument is omitted, no blank lines appear before the footer. Two blank lines appear before the footer by default.

Paging

.op Odd page

Forces the next page to be an odd numbered one.

.pa # Page

Sets the number of the page according to the argument.

Horizontal Position

.hp # Horizontal position

Sets the current horizontal position, reports warning if position is already passed. Omitting the argument sets the the horizontal position to column 1.

Diversions

Diversions handle text in both directions: text can be transferred out to a specified file or inserted from a specified file. There are six commands which deal with diversions. (Note: the term "d_name" stands for diversion_name in the following explanations.)

Output Diversions**.so filename**

All following text is output to the specified file until the end of the diversion. This command can be used throughout the body of the document, and all additions are appended to the specified file. It is best used as a repository for oft repeated text.

.di d_name

The same as ".so" except that a temporary file is created. Thus two people, working in the same directory, can be directing output to a diversion called "test" and not address the same file. Each person has a unique temporary file.

.ed Ends diversion.**Reading In Diversions .ri filename**

Reads in from the specified file until the end of the file is reached.

.rd d_name

Reads from the d_name and does not do a truncate, thus enabling multiple reads.

.gd d_name

Reads from the d_name and truncates the diversion.

Comments**.co**

The text following is interpreted as comments. The command is most often used to comment user-defined macros in a macro file.

Special Printing

The following commands were written to take advantage of the various fonts available on the different Spinwriter thimbles. The device type must be specified, either on the command line or in the input file before the text. If you specify the **heath** terminal and you are using a Zenith 2-19, all special printing will be displayed as reverse video.

format

If you do not have the appropriate printer or terminal do not specify these commands.

.dv Device type

Sets the device type, used in the underlining and the bold commands to determine the correct escape sequences.
(Recognizes the same devices as the command line.)

.ul Underlining

Turns on underlining and continues to underline until turned off.

.nu No underlining

Turns off underlining.

.bo Bold

Turns on bold type face.

.nb No bold

Turns off bold.

.it Italics

Prints capital letters in italics. (If using the Light Italic thimble.)}

.ni No italics

Turns off italics}

.sh Shadow

Turns on shadow printing.

.ns No shadow

Turns off shadow printing.

Breaks

Page breaks are caused by: bp, dv, lm, m1-4, pa, pl and op.

Line breaks are caused by: ad, br, ce, di, ed, et, fi, gd, in, ll, ms, na, ne, nf, pi, rd, rj, so, sp, ta, ti, and tr.

These commands do not cause any form of break: bo, cb, co, ef, eh, fh, fo, he, hp, it, nb, nc, ni, ns, nu, of, oh, sh, and ul.

EXAMPLES

Macro file

A typical macro file might contain the following macro:

```
#contents(title)[.ll 65
    .pl 66
    .lm 5
    .ml 4
    .m2 2
    .sp 6
    .ce
    title
    .ad
    .he !title!!Introduction!
    .m2 3
    .m4 2
    .fo !copyright(c) 1983 CRDS!\\p!!]
```

The macro, `@contents()`, sets the line length, page length, and the amount of lines around the header and footer. In addition, it spaces down six lines and puts out the centered title, **Learning About UNOS** as specified in the macro. The page header also contains the title, at the left margin, and the word **Introduction** at the right margin, and the footer with the copyright statement at the left margin and the current date at the right margin.

DIAGNOSTICS

Diagnostics usually refer to source line numbers.

ALSO SEE

User's Guide to UNOS
UNOS Text Processing

NOTES

Input lines that begin with either a "." or "@" are treated as if they were **format** commands. Use either a leading white space

format

or the translate (.tr) command to translate the special character to text.

Alphabetical List of all Commands

.ad	adjust	.m4	# blank lines after footer
.bo	bold	.ms	skips
.bp	break page	.na	no adjust
.br	break line	.nb	no bold
.cb	change bar	.nc	no change bar
.ce	center	.ne	need
.co	comments	.nf	no fill
.di	diversion (tmp file)	.ni	no italics
.dv	device type	.ns	no shadow
.ed	end diversion	.nu	no underlining
.ef	even footer	.of	odd footer
.eh	even header	.oh	odd header
.et	end translate	.op	odd page
.fh	forced header	.pa	page
.fi	fill	.pi	picture
.fo	footer	.pl	page length
.he	header	.rj	right justify
.hp	horizontal position	.rd	reads from diversion
.gd	reads/deletes diversion	.ri	reads from filename
.in	indent	.sh	shadow
.it	italics	.sp	space
.ll	line length	.ta	tabs
.lm	left margin	.ti	temporary indent
.m1	# blank lines before header	.tr	translate
.m2	# blank lines after header	.ul	underlining
.m3	# blank lines before footer		

frecv

frecv allows a remote CRDS machine to receive a file from a host

FORMAT

frecv *file_name*

DESCRIPTION

frecv assumes that you are invoking it on a remote machine which is connected via the **term** command to the host machine. It sends the appropriate control character sequences which cause **term** to enter its transmit mode (**term** invokes **mnet**). **frecv** invokes **mnet** and passes it the *file_name* with your login name appended to it. (This is to ensure against accidental over-writes). Status of the file transfer is logged in **/tmp/__fnet__log**.

OPTIONS

none

EXAMPLES

To send a local file (*foo.c*) to a remote CRDS system, type
\$frecv foo.c.

frecv will copy the file to the remote CRDS machine's current working directory, after appending the sender's login name to the file.

DIAGNOSTICS

Error messages are reported to **/tmp/__fnet__log**.

ALSO SEE

ftrans allows a remote CRDS machine to send a file to a host
mnet transmits a file via serial tty ports
mnet receives a file via serial tty ports
term asynchronous terminal emulator

frecv

NOTES

none

fscheck

fscheck checks and fixes file and directory structures

FORMAT

fscheck [option=value][-flag] /dev/rname

DESCRIPTION

fscheck checks the directory structure and the block allocation on the specified device. **fscheck** reads the entire disk and detects ordinary data blocks that are unreadable due to permanent disk errors. If it passes a disk, the disk should have no problems when mounted.

fscheck puts all orphaned files in **/lost+found** of the particular file system which may be **/src/lost+found** or **/usr/lost+found** etc.

You can specify the options before or after the disk name, and you can specify multiple disk names.

OPTIONS

b=range Prints file names to which specified blocks belong.

range is a comma-separated list of numbers and number ranges. For example, "**b=100,300-400**" means block 100 and all blocks from 300 to 400.

f=range Prints file names corresponding to specified file numbers. **range** is the same as for *b=*.

-fix Fixes all fixable errors. With this option, **fscheck** always rewrites the bit maps and updates the file descriptors and directories as necessary.

-fixmodes Inserts default group modes and **x** permission on directories. UNOS Rev. 5 supports groups and requires **x** access to search a directory; this option to **fscheck** can be used to ease the transition. It copies the "others" mode to the "group" mode for all files, and sets **x** for directories. It implies *-fix*.

fscheck

-silent Suppresses error messages for all fixable conditions.

-stats Prints out statistics. **fscheck** counts the number of instances of each of these items (see **NOTES** for a complete description of the four normal file types):

files used number of file descriptors in use.

files free number of file descriptors available (this number with the number of files used should total the number of files specified when the file system was created).

four normal file types
 number of small, large, huge, and gargantuan files

four distinct contiguous file types
 number of small, large, huge, and gargantuan contiguous files.

contig files always 0.

buffered dev/unbuffered dev
 number of real devices, buffered and unbuffered.

event files number of eventcounts.

directories number of directories. Also counted as files.

large directories
 number of directories that have more than 8 blocks.

blocks used/free
 number of blocks used and the number free; together should add up to the number of blocks available when the file system was created.

indir blocks number of indirect blocks.
 The blocks that point to data blocks for

fscheck

large, huge, and gargantuan files, and for large, huge and gargantuan contiguous files.

dblind number of double indirect blocks.
The blocks that point to single indirect blocks for huge and gargantuan files, and huge and gargantuan contiguous files.

treble indir blocks number of triple indirect blocks.
The blocks that point to double indirect blocks for gargantuan files and gargantuan contiguous files.

directory blocks number of blocks allocated to directories.

dir ind blocks number of blocks that point to directory blocks for large directories, and large contiguous directories.

-test Test block. Examines the test block. If the file system is marked as unmounted, **fscheck** assumes the file system has been correctly shutdown. If it is marked as mounted, **fscheck** will assume that the file system was brought down abruptly. (For more information about the ways that file system can be shut down, see the discussion of auto-power up/down in the *UNOS SYSTEM CONCEPTS MANUAL*.)

-verbose Verbose Prints out the program's progress.

Messages

While the errors vary in seriousness, most are fixable with the **-fix** option. Messages beginning with ******* indicate errors that should be fixed before the file system is used, because they could make things worse. Other messages indicate conditions that are not likely to cause any ill effect beyond wasting space.

***** Root is not a directory!**

The file descriptor for the root directory doesn't have the

fscheck

directory attribute. `fscheck -fix` turns on the directory flag.

***** Root directory (0) is invalid type %d.**

***** Bad block file (1) is invalid type %d.**

The special file named has an incorrect type. It must be either a SMALL or a LARGE file. `fscheck -fix` destroys and recreates the file with a type of SMALL.

***** Root directory (0) not marked busy in bitmap.**

***** Bad block file (1) not marked busy in bitmap.**

These two special files should always be specified as busy. `fscheck -fix` merely allocates the affected file in the bit map.

***** Root directory (0) has refcnt = %d, should be zero.**

***** Bad block file (1) has refcnt = %d, should be zero.**

The reference count in the file descriptor is not zero. There should be no references to the root, which would imply a recursive directory structure, or to the bad block file, which is reserved. `fscheck -fix` fixes either problem by clearing the reference count.

***** File descriptor %d garbled.**

The file descriptor contains an incorrect file number, negative size, or some other inconsistency that probably indicates it has been overwritten with data of some other type. This problem is likely to affect all the file descriptors in a disk block. `fscheck -fix` destroys and reinitializes the file, and marks it free in the bit map.

***** Bad type %d: '%s'.**

The file type in the file descriptor is an illegal type (not one of the values currently recognized). `fscheck -fix` fixes this by destroying the file and marking it free in the bit map.

***** Directory has type %d: '%s'.**

An error was found when type checking. The file is incorrectly designated as a directory when it actually is a device or an eventcount. Only large and small files can be directories. `fscheck -fix` fixes this by turning off the directory attribute of the file. The type number means: verbatim{number type -----
----- 0 unused 1 small file 2 large file 3 contig 4 unbuffered
device 5 buffered device 6 huge file 7 eventcount 8 gargantuan

file 9 small contiguous file 10 large contiguous file 11 huge
contiguous file 12 gargantuan contiguous file}

File allocated but type is UNUSED: #d.

The bit map shows that this file is used, but the file descriptor contains the type UNUSED. The file descriptor has not been filled in by the process of creating a new file, so this is not a valid file. **fscheck -fix** fixes this by marking the file free in the bit map.

***** File not allocated but type is not UNUSED: '%s'.**

The bit map shows that this file is free, but the file descriptor appears to describe a valid file. **fscheck -fix** fixes this by marking the file busy in the bit map.

Allocated but not referenced: #d.

Although the bit map shows that the file is used, and the file descriptor is valid, the checker could not find a directory entry for the file. **fscheck -fix** fixes this by creating a directory entry in **/lost+found** with a name made from the file number, like file_123.

***** Refs says %d, seen %d: '%s'.**

Refs says %d, seen %d: '%s'.

The reference count in the file descriptor does not correspond to the number of directory entries that reference the file. As long as the file descriptor number is larger than the actual number of references, this is not a serious error. If the file descriptor number is smaller, the message is preceded with *******. **fscheck -fix** fixes this by putting the actual reference count in the file descriptor.

***** Reference to file of type UNUSED in directory entry '%s'.**

The file descriptor is referenced in a directory but its type is UNUSED, which indicates that it is not a valid file. Any of the problems that **fscheck** fixes by destroying and freeing a file descriptor may produce this message for each reference to the affected file. **fscheck -fix** fixes this by deleting the directory entry.

fscheck

*** Invalid file number in directory entry '%s'.

A directory entry with the given name refers to a file number which is outside the valid range. This could be a negative number, too large a number, or a reserved file number (0 and 1). **fscheck -fix** deletes the directory entry.

*** Invalid file name in directory entry '%s'.

A directory entry contains a name that can not be used within UNOS, because the file system attaches some special meaning to it. The invalid names are '.' and '..', a zero-length name, and anything containing a slash ('/'). **fscheck -fix** fixes this by deleting the directory entry.

'/lost+found' isn't a directory.

'/lost+found isn't a small or large.

These messages can only occur when the **-fix** option is used. Either indicates that a file already exists with the name **fscheck** uses for a directory of orphaned files. **fscheck** can't fix this, but deleting **/lost+found** and running **fscheck -fix** again should get past the problem.

Can not put file in /lost+found.

This message can only occur when the **-fix** option is used. It indicates that the orphaned file mentioned in the immediately preceding "Allocated but not referenced" message could not be put in the directory **/lost+found**. One likely reason for this error is that the file system has no free blocks left with which to extend the directory. **fscheck** can't fix this, and it leaves the file in the "Allocated but not referenced" state. To get around this error, delete some files or remove some names from **/lost+found**.

*** Bad block 0x%x: '%s'.

The block number in the file is not within the legal range: a low number, in the system area, or a large number, too large for the file system and thus outside the legal range. **fscheck -fix** fixes this by clearing the bad entry in the file's block map or indirect block.

*** Bad extent at %d (location): '%s'.

*** Bad extent at %d (length): '%s'.

fscheck

The extent in the contiguous file named is not within the legal range. `fscheck -fix` fixes this by truncating the file to remove this and subsequent extents.

***** Subextent %d garbled: '%s'.**

The contiguous file named has a missing extent. `fscheck -fix` truncates the file.

***** Subextent %d size wrong: '%s'.**

The count of blocks pointed to by an indirect block of a contiguous file is incorrect. `fscheck -fix` fixes the count.

***** Subextent %d of zero length: '%s'.**

A contiguous file has an indirect block that doesn't point to any blocks. `fscheck -fix` frees the block.

***** System block %D should be marked busy.**

The beginning of the disk is the system area and contains the bit map and the file descriptors. These blocks must be marked as busy in order not to be allocated to other files. `fscheck -fix` merely marks the block busy.

***** Block %d is used but not marked busy: '%s'.**

The bit map indicates the block is free, but it is referenced in a file. This is a serious problem because the block will be further allocated. `fscheck -fix` fixes this by marking the block busy.

Block %D marked busy but not used.

Blocks %D to %D marked busy but not used.

The free block bit map indicates that the block is used, but there is no corresponding file. This is not a harmful error, rather it is wasteful of space. `fscheck -fix` marks the block free.

***** Block %d seen in a file more than once: '%s'.**

Also seen in '%s'.

The block is shared by two or more files, or appears two or more times in a single file. `fscheck -fix` removes all but one of the references to the block, thereby making the file system consistent, if not right. The remaining reference to the block should be removed manually, by deleting the file named in the second line of this message.

fscheck

Unreadable block %d in '%s'.

***** Block %d past end of partition in '%s'.**

The block identified caused an I/O or EOF error when **fscheck** tried to read it. The block does not contain any of the file system structure, or **fscheck** would consider the error fatal (a different message; see below). **fscheck -fix** does not attempt to fix this error.

File has no group modes set: '%s'.

This message can only occur when the **-fixmodes** option is specified. The file named has no modes set in the "group" field (which is the expected state for files created before Rev. 5, which implements groups). **fscheck** fixes this by copying the "others" field into the "group" field.

Revision 4, set_uid updated: '%s'.

This message can only occur when the **-fixmodes** option is specified. The file named has a bit turned on that meant SETUID in Rev. 4 (an undocumented feature). **fscheck** moves the bit to the proper place for Rev. 5.

Directory has no X permission: '%s'.

This message can only occur when the **-fixmodes** option is specified. The directory named does not have **x** permission set in any of the mode fields (owner, group, others). **fscheck** sets **x** in all three fields.

File system marked as mounted ...

This message occurs when **fscheck** is run on a file system actually mounted, or one that was mounted when the system crashed. It may be fixed by running **fscheck** on it with the **-fix** option. Such a file system could be damaged. It can only be mounted with the **-crashed** flag. It will then be read only.

EXAMPLES

All fixable problems are silently righted. Unfixable conditions are not reported. Fix works by rewriting the bit map and the file descriptors.

```
$fscheck -s -fix /dev/rs03
```

DIAGNOSTICS

These errors occur when there is some difficulty checking or fixing the file system. When any of these is encountered, **fscheck** gives up on the current file system, and proceeds to the next one named as an argument (if any). The messages are:

fscheck: <reason> Can not open device '%s'.

Occurs if the device name does not exist or if the user does not have access to the device. **fscheck -fix** requires write access to the disk and will fail with this message if the file system is currently mounted.

fscheck: File system header is missing or garbled.

Device used did not have proper file system information or it did not look like a file system.

fscheck: Can not allocate %d bytes for bit maps.

There isn't enough space in memory to hold the data structures needed by **fscheck**. The space required is 1/2 byte for each block on the disk plus 2.125 bytes for each file descriptor plus 2560 bytes or less for rounding up to block sizes. For a typical 20000 block partition, this comes to 16384 bytes. As **fscheck** itself is less than 50K bytes, the most likely reason for this error is that swapping is disabled and memory is filled with other processes.

fscheck: An I/O error occurred. Error reading disk.

fscheck: An I/O error occurred. Error writing disk while attempting fix.

The disk driver detected and reported an I/O error. **fscheck** stops immediately if the error occurs on a block that it needs in order to check the file system structure (i.e. file descriptor or directory block). **fscheck** reads all the allocated data blocks of the disk in order to verify that they are readable, and an error on such a block is not treated as fatal.

For this reason, and because **fscheck** reads the disk in large chunks, the kernel may print its I/O error messages for blocks which **fscheck** doesn't need. Remember, if **fscheck** itself doesn't complain about an I/O error, then it is a harmless error -- the file system may be safely used.

fscheck

ALSO SEE

makefs initializes a file system

space displays amount of free space in the file system

UNOS System Concepts Manual

NOTES

After the system crashes, you should boot from a back-up root which can be stored on another part of the disk, on a floppy or on a cartridge. The *-fix* option should only be used on the unmounted file systems. When you boot a system, the root is automatically mounted.

When **fscheck** attempts to locate a filename in a directory, it starts with the filename and works backwards to the root. It is possible to lose the path at any point, the worst case being just the file descriptor number with no accompanying filename.

The four file types described by the *-stats* file items (small files, large files, huge files, and gargantuan files) are differentiated by number of levels of indirect blocks. Their differences are not discernable to the user, but may be of interest.

- Small files do not use indirect blocks, and are limited to 8 blocks of data (normal files) or three extents (contiguous files).
- Large files use one level of indirection, giving up to 1024 blocks of data (normal files) or 192 extents (contiguous files).
- Huge files use two levels, for a limit of 131072 blocks (normal files) or 12288 extents (contiguous files).
- Gargantuan files use three levels, and are limited for other reasons to 4194304 blocks (normal files) or 786432 extents (contiguous files).

USES

The devices specified.

ftrans

ftrans allows a remote CRDS machine to send a file to a host

FORMAT

ftrans *file_name*

DESCRIPTION

ftrans assumes that it was invoked on a remote CRDS machine which is connected to the host machine via the **term** command. It sends the appropriate control character sequences which cause **term** to enter its receive mode (**term** invokes **mnetr**). **ftrans** invokes **mnet** and passes it *file_name*. *file_name* with **.remote** appended to it is passed to the **term** on the remote machine (which in turn, passes on to **mnetr**). The *.remote* appendage is to help ensure against accidental over-writes. Status of the file transfer is logged in **/tmp/_fnet_log**.

OPTIONS

none

EXAMPLES

To transmit a file to a remote machine running **term**, type
\$ftrans foo.c

The file is copied to the user's current working directory on the remote machine with the suffix *.remote* appended to it.

DIAGNOSTICS

Error messages are reported to **/tmp/_fnet_log**.

ALSO SEE

frece allows a remote CRDS machine to receive a file from a host

mnet transmits a file via serial tty ports

mnetr receives a file via serial tty ports

term asynchronous terminal emulator }

ftrans

NOTES

none

gcc compiles C programs using the Green Hills C compiler

FORMAT

gcc [options] file1..filen

DESCRIPTION

gcc compiles and links a C program using the Green Hills C compiler.

As a convention, C programs normally have a **.c** suffix. You do not need to use the **.c** suffix when you execute **gcc**. If a file name does not end in **.c** and has no other extension, **gcc** appends a **.c** to it and continues. If you execute

```
$gcc foo
```

gcc compiles **foo.c** and creates an executable file named **foo**.

gcc calls **ccom68** and **casm** which compiles the program. **gcc** also calls **ld** which links the program.

If you specify a file with a **.s** extension, **gcc** assumes the file is an assembly source file and will not call **ccom68**. If you specify a file with any other extension, **gcc** assumes the file is an object file that only needs linking. In this case, **gcc** does not call **ccom68** or **casm**.

gcc always specifies an output file name to the linker, so the output does not default to **a.out**. If the **o=** option is not used the default is the first file name given to **gcc** stripped of its suffix.

gcc creates the **.o** file in the working directory even if the source file is in another directory. This is consistent with the Pascal driver and with what the build program expects. **gcc** eliminates duplicate **.o** files from its link argument list.

gcc recognizes many of the linker options, including **ssize=**, **-sbt**, **-lun0** and **-uv** and the very primitive **tbase=**, **dbase=** and **sbase=**. **gcc** passes any number of arguments to the linker, not just a maximum of 128.

gcc

gcc puts **/lib/c68rt0.o**, that sets up the arguments to **main** and calls **main()**, at the front of your program.

gcc always calls the Green Hills version of the UNOS library, **/lib/unos.lib**. If you specify any libraries with the **l=libraries** option, **gcc** scans the UNOS library first.

OPTIONS

-autoreg Enables the automatic allocation of local variables to registers by the compiler. This option is on by default. Disabled it with **-noautoreg**. If disabled, only variables explicitly declared **register** may be placed in registers.

-c Compiles only. Stops after calling **casm**. The output is a **.o** file.

-coff Instructs **gcc** to use COFF format for output files by invoking **casm** and **ld**. The **-coff** option is on by default in Release 7 of UNOS.

-C Includes comments in the preprocessor output. Only meaningful if **-E** or **-P** is specified.

d=defn

D=defn Provides a means of passing a define to the preprocessor.

-Dname Defines *name* to the preprocessor with the value 1. This is equivalent to putting **#define name 1** at the top of the source file.

-Dname=string Defines *name* to the preprocessor with the value *string*. Using this option is equivalent to putting **#define name string** at the top of the source file.

-Dunos Sets definitions specific to unos operation.

-Dunix Sets definitions specific to unix operation.

-DM68000 sets definitions specific to the Motorola 68000 processor.

- D___GREENHILLS sets definitions specific to the Green Hills compiler. `lm10`
- E Runs the C preprocessor phase only, output from the preprocessor is written to standard output.
- f Uses the floating-point processor. When this option is specified, the compiler generates code to access the Sky FFP (Fast Floating-point Processor) for arithmetic on float and double types. If any `.o` file is compiled with this option, the link step must also be used to get a different standard library `/lib/unos.f.lib` that includes code to initialize the Sky FFP.
- i A null flag that will not be rejected but won't do anything either (for compatibility with UNIX makefiles).
- Istring Searches for include file names which do not start with `/` in the directory *string* before a standard list of directories. Multiple `-I` options can be specified. They will be searched in the order encountered.
- llibrary
- l=library* Adds libraries. The user can specify the pathname of additional libraries (to be searched in order) other than those `gcc` usually uses.
- lun0 Denotes the default linker option. Prior to Rev. 5, the default was `-no_lun0`. When this option is on, logical unit numbers and UNIX fildes numbers are the same. To override the default, specify `-no_lun0`.
- n A null flag that will not be rejected but won't do anything either (for compatibility with UNIX makefiles).
- nocoff Instructs `gcc` to produce UNOS Release 6 (*rel* format) object modules by calling `asm` and `link`.
- ooutput file
- o=file* Names the output file. Uses this option to specify a file other than the default output.

gcc

- O Causes loop optimizations. The compiler already performs optimizations, so this flag has no affect. It will not be rejected, for compatibility reasons.
- popalot Forces the compiled code to clean up the stack often. Normally, the compiler tries to save code by only cleaning up the stack when necessary or at the end of a procedure.
- probe Generates a stack probe instruction on procedure entry. This enables automatic stack growth for UNOS programs. This option defaults to on (disable it with **-noprobe**).
- P Resembles **-E** except that output is written to a file with the same name as the input file with the extension changed to **.i**.
- S Produces assembly language instead of object code. With this option, **gcc** does not run through **casm**; it stops after **ccom68**. The output has a **.s** extension and is assembly language source.
- Uname Undefines the predefined preprocessor symbol *name*. This is equivalent to putting **#undef name** at the top of the source file.
- w Suppresses warning messages in error output.
- X<number> Passes an -X option directly to **ccom68** See **ccom68** command page for a description of the -X options.
- Z<number> Passes a -Z option directly to **ccom68**. See the **ccom68** command page for a description of the -Z options.

RETURNS

gcc returns a correct exit code (for the benefit of the **build** program). A 0 returns if it succeeds or an appropriate error code (whatever code is returned by one of the steps it execs).

NOTE: **ccom68** only returns TRUE/FALSE, so **gcc** returns 1 if it returns FALSE.

If any step cannot run to normal completion (i.e. fork failure, exec failure, step suspends, or is killed), **gcc** stops immediately, prints at least one message and returns a non-zero error code.

EXAMPLES

In this example, the compiler compiles lines found between **#ifdef DEBUG** and **#endif** which is a common way of including optional debugging statements in a program.

```
$gcc d=DEBUG program.c
```

file.o is the unlinked binary object file.

```
$gcc -c file.c
output to file.o
```

The user can define the preprocessor symbol.

```
$gcc d=name file.c
```

The user can define the symbol and use its value.

```
$gcc d=name=value file.c
```

The file becomes the executable program.

```
$gcc o=file file.c
```

Assembly language is output to **file.s**.

```
$gcc -S file.c
```

Preprocessor output is placed in **file.i**.

```
$gcc -P file.c
```

DIAGNOSTICS

The diagnostics produced by **gcc** are self-explanatory.

ALSO SEE

cc compiles a C program using the Whitesmiths C compiler.

cocom68 compiles C programs

ld binds object modules

link binds object modules

wcc compiles a C program using the Whitesmiths C compiler.

gcc

C Reference Manual by D.M. Ritchie
Programming in C -- A Tutorial by B.W. Kernighan

NOTES

See the C release notes for information about C implementation on UNOS.

The "\<NL>" syntax for line continuation does not work with **gcc**, though it does for **wcc**.

Note that the object file format has changed for UNOS Revision 7.0.

The following **ccom68** flags are on by default for the **cc** and **gcc** commands. Parts of UNOS code use the special -U and -D definitions.

-coff	-X102
-ga	-X122
-X12	-X138
-X14	-Uunix
-X16	-Dunos
-X22	-DM68000
-X81	-D__GREENHILLS
-X99	

See Appendix A of the *UNOS Developers Manual* for a table comparing the flags used by **gcc** and **cc**.

USES

with *rel* format object modules
/lib/unos.lib
/lib/unos.archive
/lib/c68rt.o

with COFF
/lib/libunos.a
/lib/c68rt0.o

genlib

genlib generates library entries from object programs and archives

FORMAT

```
genlib library_file input_file1...input_filen
```

DESCRIPTION

genlib installs new subroutines in a library or creates a new library of subroutines.

A **library_file** is simply an ASCII file containing names of external definitions and pathnames of files that resolve those externals. An **input_file** can be a UNOS object file or an archive file containing object files. **genlib** automatically recognizes archive files and creates entries for each object file in the archive. The linker knows how to search libraries to resolve external references and to link the necessary files. The linker also knows how to extract object files from the archive.

OPTIONS

none

EXAMPLES

To install the file, **x.j**, in an existing library, type
`$genlib library x.j`

To replace the file **x.j** with a newer version, type
`$genlib library x.j`

To create the library and install all **.j** files in the current directory into the library, type
`$genlib library *.j`

To insert or replace all object files in the archive file **archfile.a** into the library, type
`$genlib library archfile.a`

genlib

DIAGNOSTICS

A warning message is generated if a global symbol is duplicated between input files or between an input file and an object file with a different name in the library:

CAUTION: duplicate symbol <name> in <file1> and <file2>.

ALSO SEE

link binds object modules

NOTES

A library file consists of a set of lines listing global names and the object file that resolves them. There is one line for each object file and it has the following format:
gname1 gname2 ... gnamen = /pathname/oname

If the object file is in an archive the format is
gname1 gname2 ... gnamen = /pathname/archname\$oname

Pathnames in the library are always absolute. If a relative pathname is given to **genlib**, it automatically generates an absolute pathname to the current working directory. Globalnames are obtained from the relocation information in the object file header. If the object was generated by the C compiler, all global names begin with an underscore ('_').

If an existing subroutine is modified without adding or removing entry points, it is not necessary to update the library file. Since library files are text and can be edited, this is the only way to remove an entry from the library short of recreating the entire library.

USES

All specified files.

.genlib.pid In the same directory as the library file.

getopt parses a command line

FORMAT

```
set -- 'getopt opt_template $*'
```

DESCRIPTION

getopt formats options in command lines for parsing by shell procedures. It uses **opt_template** to determine what values it accepts as valid options. You can find details of the layout of the template in the **getopt()** section of the *UNOS Subroutine Manual*. **getopt** uses double dashes in the string it returns to mark the boundary between options and files. These double dashes appear as part of the command input; they are not, however, mandatory.

OPTIONS

None

EXAMPLES

The shell script that follows parses a command line for which four options, a, b, c, and d, can be specified followed by any number of file specifications. In addition, options c and d carry arguments.

```
set -- 'getopt abc:d: $*'
if [ $? != 0 ]
then
    echo "foo -a -b -carg -darg files"
    exit
fi
for i in $*
do
    case $i in
        -a) AFLAG=$i;shift;;
        -b) BFLAG=$i;shift;;
        -c) CFLAG=$i; CARG=$2; shift 2;;
        -d) DFLAG=$i; DARG=$2; shift 2;;
```


getopt

```
        --) shift; break;;
    esac
done
echo $AFLAG
echo $BFLAG
echo $CFLAG
echo $CARG
echo $DFLAG
echo $DARG
for i in $*
do
echo $i
done
```

When this shell script is placed in a file and executed, it produces output of the type that follows:

```
shell> foo -a -b -c c1 -d d1 file1 file2 file3
```

```
-a
-b
-c
c1
-d
d1
file1
file2
file3
```

DIAGNOSTICS

Self-explanatory

ALSO SEE

ksh, **rsh** shell, the standard/restricted command programming language

getopt() gets option letter from argument vector, for UNIX style argument parsing

NOTES

none

USES

nothing

hexdec

hexdec displays the contents of a file in various formats

FORMAT

```
hexdec [-c!count=][-r!radix=][size=][start=]  
[file_name]
```

DESCRIPTION

hexdec displays the contents of the specified file in various formats. The default is to display onto standard output 16 bit hexadecimal words to the right with their ASCII interpretation to the left. However, various other display formats can be selected with the options (explained in the next section).

OPTIONS

-c,count= Number of bytes to display in either hex decimal or octal. The default is to display the entire file.

-r,radix= Displays radix:
h - Hexadecimal (the default)
o - Octal
d - Unsigned Decimal

size= Displays size:
s - Short (16 bits) (the default)
l - Long (32 bits)
b - Byte (8 bits)

start= Starting byte count in either hex decimal or octal. The default is 0.

file_name Name of file to display. The default is stdin.

hexodec

EXAMPLES

Here is an example of using **hexodec** on this file:

```
$hexodec kmail
```

```
B0      : 200A 0A0A 6865 786F 6465 6320 5B73 697A      ....hexodec.[siz
00010:  653D 5D5B 2D72 2172 6164 6978 3D5D 5B2D      e=][!radix=][-
00020:  6321 636F 756E 743D 5D5B 7374 6172 743D      c!count=][start=
00030:  5D20 5B66 696C 655F 6E61 6D65 5D0A 0A0A      ].[file_name]...
00040:  4445 5343 5249 5054 494F 4E3A 0A0A 6865      DESCRIPTION:...he
```

The first field is the address (in HEX) of the first object of the line. When block boundaries (512 bytes) are crossed, the block number is displayed instead. The second group of fields is the contents of the file displayed as specified by the size and radix option. The display on the right is the ASCII representation of the file data. If the ASCII representation is "non-printing", a "." is displayed.

```
$hexodec size=b kmail
```

```
B0      : 20  0A  0A  0A  68  65  78  6F  64  65  63  20  5B  73  69
          .   .   .   .   h   e   x   o   d   e   c   .   [   s   i
00010:  65  3D  5D  5B  2D  72  21  72  61  64  69  78  3D  5D  5B  2D
          e   =   ]   [   -   r   !   r   a   d   i   x   =   ]   [   -
00020:  63  21  63  6F  75  6E  74  3D  5D  5B  73  74  61  72  74  3D
          c   !   c   o   u   n   t   =   ]   [   s   t   a   r   t   =
```

The display changes when the size is specified as byte. The ASCII characters are displayed UNDER their corresponding hex (decimal or octal) interpretation.

DIAGNOSTICS

none

ALSO SEE

nothing

hexodec

NOTES

none

USES

nothing

hist

hist constructs a histogram of function usage

FORMAT

```
hist  
[-sys][-v][bar=#][o=file][pid=#][usym=file][ssym=file][logfile]
```

DESCRIPTION

hist reads the **logfile** produced by **logger** and constructs a histogram of function usage. If no logfile is specified, it reads standard in.

hist is an internal command. It is intended to be used only by the **profile** command; the interface is subject to change.

OPTIONS

-sys Includes supervisor mode entries for the specified pid as well a user mode entries. If no pid was specified, this option is meaningless.

-v Verbose mode. Prints all entries selected from the **logfile** as they are read.

The format is

entry number; addressing mode(S or U); pc;
followed by the function if it can be determined. If no pid was specified, all user mode entries have the pid number instead of a function.

bar=# Sets the scaling factor for the histogram bars to the specified number. The default value is 10. **Bar=0** suppresses printing of the bars, showing only the function name and the reference count.

o=file Allows the output to be put into the file [file].

pid=# Selects only entries only with a matching pid. Selects only user mode entries unless **-sys** is used (q.v.).

hist

usym=file Specifies the file to read for symbols to resolve user mode addresses. This would normally be the (unstripped) object file of the program that ran as pid. If no pid is specified, this option is meaningless.

ssym=file Uses this file to resolve supervisor mode addresses, instead of the default file **/sys/unos**.

EXAMPLES

The following example shows all entries in the **logfile**.

Supervisor mode references are resolved with **/sys/unos**. User mode references show pid number.

```
$hist -v logfile
```

The next example prints user and supervisor mode histograms for pid 123, which was a process running the program *testprog*.

```
$hist -sys pid=123 usym=testprog logfile
```

DIAGNOSTICS

hist generates an error message if it cannot open the necessary files.

ALSO SEE

logger collects timing information

profile produces a function histogram

NOTES

In order to produce a histogram, you must have a symbol file. If you specify a pid without the accompanying *usym* file and without the *-sys* option, you will only get a count of the user and symbol mode references for that pid.

USES

nothing

id

id prints user and group IDs and names

FORMAT

id

DESCRIPTION

id writes user and group IDs of the invoking process to standard output. If the effective ID and the real ID differ, both are printed.

OPTIONS

none

EXAMPLES

```
$ id
```

```
uid=289 (peterpan) gid=0 (cartoonchar)
```

DIAGNOSTICS

none

ALSO SEE

nothing

NOTES

none

USES

nothing

installation

installation reads files from a tape or disk in an installation file format

FORMAT

installation [-x] [-v] [-b] [-t] [-f] [-c] [-s string]
[-recover] [bs=#] [patterns]

DESCRIPTION

installation reads a tape created by the UNIX cpio or UNOS package programs. Use it for tapes and disks created on foreign sites with UNIX tools, and for reading recent Charles River Data Systems distribution tapes or disks. Used with package, **installation** provides a way of doing system backup and files maintenance.

In normal operation, **installation** reads data from the standard input, which is assumed to be an archive in cpio format, and it restores the files found in that archive.

If the pattern option is included, **installation** only restores those files that match the file name patterns (exceptions noted below).

OPTIONS

-x Specify device node before patterns.

This lets you name a device to read from, rather than reading from the standard input.

-V Volume headers for multi-volume sets.

When an archive takes up many tapes or disks, this option expects information at the beginning of each one that instantly indicates misordered volumes.

NOTE: This option is NOT cpio compatible.

-b small buffersize (512 bytes).

Normally, **installation** expects tape or disk blocks to be 5120 bytes long (except for *bs=#* below). This option expects 512-byte blocks. Do not use this option with the *bs=#* option. Use this option to read data from a pipe, for example,

installation

when data is coming from the **tp** command, or a similar utility.

-t table of contents.

No files are extracted. Instead, the access modes, user id, group user id, size of file in bytes, date last modified and the names of the files that would have been extracted are printed to standard out.

-c Install files as contiguous.

This option tries to install all files as single extent contiguous files. If the **-c** option can not install the files as contiguous or 'chunky', (chunky is no extent smaller than 1/10 of the file), the **-c** option will create the files as a standard block file. This helps increase the I/O speed of binary files, and saves a little disk space because indirect blocking is not needed.

NOTE: This option used with **package** can de-fragment a disk by dumping the disk with the **package** command, re-creating the file system with the **makefs** command, and by using the **installation** command with this option.

-s *string* Strips the front part of the file pathname strings.

This option strips off the front of a pathname string and installs the string into the pathname of the current working directory.

For example, use this command sequence to transfer a tape dump from **/usr/src** to **/usr1/jr/src**:

```
#cd /usr1/jr
```

```
#installation -s /usr/ </dev/tape
```

Now, the files are in **/usr1/jr/src**.

-recover Recovery mode.

This mode recovers from abnormal file dumps. If a bad dump causes a **File Header Expected Error**, the **-recover** option continues trying to read the rest of the volume. Use this option to recover from a bad backup or to recover a lost volume, for example, the third floppy of 10, the first one, etc.

installation

NOTE: *-recover* option does not handle I/O errors. For an I/O error on a floppy, use the **debe** with the *-noerror* option to read the floppy to the disk. Then, rewrite a new floppy from the disk also with the **debe** command. Then, you can use the *-recover* option with the **installatin** command in case the filled-in bad block was in a header.

1/4" streaming tapes CANNOT recover from a bad block. Tapes must read every block to know where they are, so you cannot 'seek past' the error spot. This situation is a hardware limitation of the tape, so make backups alternating between two or more tapes.

bs=# Set the I/O buffer size.

This option reads from 1/4" tapes to allow the tape to stream. The memory is rounded to the nearest 5120-byte block for *cpio* compatibility. You can specify # in bytes, #k for 1024-byte chunks, or #b for blocks or 512-byte chunks, for example, *bs=100k*, *bs=100b*, *bs=25000*.

-f extracts all but (patterns)

The patterns you specify are the files NOT copied.

EXAMPLES

This example installs any files from the tape */dev/rmt0* that match the pattern **ter**.

```
#installation -t -x /dev/rmt0 '*ter*'
```

```
100606 88, 0 18625 bytes Mon Feb 13 18:48:49 1984 /src/ter/T
100606 18, 0 451 bytes Fri Aug 10 09:53:59 1984 /src/ter/M
100606 101, 0 3079 bytes Wed Jan 4 10:20:30 1984 /src/ter/N
100666 88, 0 17817 bytes Wed May 9 12:15:39 1984 /src/ter/A
100666 88, 0 16626 bytes Thu Aug 23 11:56:54 1984 /src/ter/H
```

DIAGNOSTICS

Bad flag: flag

Options were incorrectly specified.

installation

Too many patterns!
Try several extractions.

Bad magic!
Not a cpio tape.

-f requires patterns
Option error.

Can't open device for input
The device (-x) could not be opened.

Fatal read error on input!
problems with the media or device.

Wrong volume: found x, wanted y
Mis-ordered volumes.

Not a volume header!
No volume header was found.

Trouble reading Volume Header!!
The volume header contained an error.

write error on output file!
The extracted file can not be written.

Out of space on destination file system
Need more space in the filesystem.

File header expected, didn't find one!
An error in the archive.
The *-recover* option may be used to recover from this.

can't link file1 to file2, extracting
The link could not be made.

cannot create directory
Space or permission problems.

Insufficient memory --
Some link information lost.

installation

Cannot stat file

No file or bad access permissions.

ALSO SEE

nothing

NOTES

If you are backing up or restoring, do so as admin. Otherwise, when you restore the file system, you will lose the files for which you did not have read permission.

USES

nothing

iostats

iostats executes a command and displays I/O statistics

FORMAT

iostats [-erase][-rewind][-unload] devicename [command]

DESCRIPTION

iostats executes a UNOS command and displays changes in the device driver statistics table caused by the command. Without a command, **iostats** displays cumulative I/O statistics (accumulated since system boot).

iostats is useful when copying from disk to tape, because the retry count can be an early indication that the tape medium is wearing out.

OPTIONS

- erase* Erases a streaming tape cartridge (1/4" tape) or reel-to-reel tape (1/2" tape) before executing the command line.
- rewind* Rewinds a streaming tape cartridge (1/4" tape) or reel-to-reel tape (1/2" tape) after executing the command line.
- unload* Rewinds and unloads (for removal) a reel-to-reel tape (1/2" tape) after executing the command line.

EXAMPLES

Prints the cumulative statistics for the CC1 driver.

```
$iostats /dev/rcq0
```

Erases the cartridge in the streaming tape drive, copies a disk partition onto the tape, and rewinds the tape. Then, prints the statistics for this operation.

```
$iostats -erase -rewind /dev/rct0 debe if=/dev/rcq2  
of=/dev/rct0 bs=100k -pg
```

iostats

DIAGNOSTICS

iostats reports command execution failures.

ALSO SEE

nothing

NOTES

Not all device drivers keep statistics. The CC1 driver keeps a separate set of statistics for each CC1 card (and SASI bus) in the system. Some 1/2" tape subsystems are attached using the Tape Master controller and driver, which does not keep statistics and does not respond to the options above.

The device name given to the **iostats** command should be relevant to the command that **iostats** is to execute. This is particularly necessary for operations involving the streaming tape drive since the tape statistics get updated only on a controller error, or by running **iostats**.

Output from **iostats** includes some or all of these items:

Number of transfers

How many I/O or **spfun** requests have entered the driver queue.

Average depth of queue, Maximum depth of queue

How many requests are waiting on the queue at one time. Maximum is always since boot, not during command line.

Number of sectors read, Number of sectors written

Transfers begun, including failures and retries. Sector size is a property of the device, typically 512 bytes.

Number of errors

How many requests failed because of device errors.

Number of retries

How many times a device error resulted in trying the device operation again (whether or not it eventually worked).

Number of ECC corrections

How many times the device corrected a data read error, using the ECC (error correcting code) which is appended when the data is written. Not all devices report corrections.

Number of spinwaits, Average loops per spinwait

Measure of time driver spent waiting for a device response that was supposed to be very quick.

Number of tape re-operations

How many times the device had to reread or rewrite a block before getting it right.

Number of tape rocks

How many times the device had to stop and restart the tape, because UNOS wasn't keeping up with the flow of data. Not reported for 1/2" tapes.

USES

nothing

ipccstat displays status of IPC queues

FORMAT

ipccstat [-all] [-long] [key=#] [qid=#]

DESCRIPTION

ipccstat displays information on the state of active IPC queues. One line of information is displayed for each queue unless the *-long* option is used, in which case three lines are displayed for each queue. Here is a description of each field:

NO The number of the queue slot occupied by the queue. Slots are numbered from 0 to one less than the maximum number of queues.

KEY The key of the queue.

MSGS The number of messages currently on the queue.

BYTES The number of message bytes currently on the queue.

ALLOWED The maximum number of message bytes allowed on the queue.

MODE The access modes of the queue. Displayed as rw rw rw for owner, group, and default respectively.

OWNER The owner of the queue.

GROUP The owning group of the queue.

Second line (with *-long* option):

SEQ The sequence number of the queue slot. This indicates how many times this slot has been used.

SPID The process id of the last process to place a message on the queue.

RPID The process id of the last process to receive a message from the queue.

ipccstat

CREATOR The creator of the queue.

CGROUP The creator's group.

Third line (with *-long* option):

CTIME The time of the queue's creation.

STIME The time of the most recent send to the queue.

RTIME The time of the most recent receive from the queue.

OPTIONS

All options may be abbreviated to their first letter.

-all Show status of all queue slots, not just active ones. Inactive slots have a key field of 0.

-long Display information in long form.

key=# Only display information for queue whose key matches the number.

qid=# Only display information for queue whose qid matches number. If both *key* and *qid* are used *key* takes precedence.

EXAMPLES

```
$ipccstat
```

NO	KEY	MSGS	BYTES	ALLOWED	MODE	OWNER	GROUP
0	47	1	4	16384	rw rw rw	wjr	system
1	23	1	4	16384	rw rw rw	wjr	system

```
$ipccstat -l
```

NO	KEY	MSGS	BYTES	ALLOWED	MODE	OWNER	GROUP
SEQ	SPID	RPID				CREATOR	CGROUP
CTIME		STIME				RTIME	
0	47	1	4	16384	rw rw rw	wjr	system
	2	3939	0			wjr	system

ipccstat

```
Jul  2 01:48:50  Jul  2 01:48:58  Jan  1 00:00:00
1      23      1      4      16384 rw rw rw wjr      system
      1      3922      0      wjr      system
Jul  2 01:45:37  Jul  2 01:45:41  Jan  1 00:00:00
```

DIAGNOSTICS

If the *key* or *qid* options are used and the key or qid does not exist, an error message will be printed.

ALSO SEE

IPC subroutines:

`ipcget()`

`ipcsnd()`

`ipcrsv()`

`ipcctl()`

`ftok()`

NOTES

You must know the key or be superuser to see the information on queues you don't own.

USES

nothing

kill

kill sends a signal to processes

FORMAT

kill [-l] [signal=#, -#, -SIGNAL] pid1...pidn

DESCRIPTION

kill sends a signal to each of the processes specified. The signal may be specified by number (e.g., **kill -2**) or by name (e.g., **kill -INT** or **kill -SIGINT**) To obtain a list of signal names, use **kill -l**. If the process is not ignoring or handling the signal, it is killed.

A process is specified by its process id number, **pid**. You can learn a process's id number by executing a **ps** command. However, in the time between executing the **ps** command and executing the **kill** command, the process(es) might have already terminated. In that case, the following message is generated
kill: process 104 does not exist

You may only kill processes owned by you, but the system administrator may kill any process.

Terminated processes (EXIT state) are unaffected by **kill**, since they are already dead. Only a process that waits for the child can clear the entry. A user program that does a fork/exec but does not ever wait for the child might cause this state.

Any process waiting for a killed process is given a return status, indicating that it was forcibly killed. If the Command Interpreter created the process, a message is printed on the terminal from which the original process was invoked.

OPTIONS

none

kill

EXAMPLES

To forcibly terminate processes 104 and 296, type
\$kill -9 104 296

To send a hangup signal to process 300, type
\$kill -1 300
or
\$kill -HUP 300

To obtain a list of the symbolic signal names, type
\$kill -l
HUP INT QUIT ILL TRAP IOT EMT FPE
KILL BUS SEGV SYS PIPE ALRM TERM

DIAGNOSTICS

kill: access violation for process <#>
kill: process <#> does not exist

ALSO SEE

ps displays process status
suspend halts execution of a process

NOTES

none

USES

nothing

ld links editor for common object files

FORMAT

ld [*options*] *filenames*

DESCRIPTION

ld merges common object files. It handles relocation, resolves external symbols, and, in order to support symbolic debugging, manipulates symbol table information. The resultant common object file is written by default to the file *a.out*. Unless errors occur during the invocation of **ld**, the output file produced is executable by default. Any input file in the list of *filenames* not of common object format is taken, by **ld**, to be an archive library or a text file comprising **ld** directives. If a file is determined to be a library, it is searched once at the point it is located in the argument list. Those routines which resolve undefined external references, and only those routines, are loaded. The procedure in determining which routines need to be loaded involves the multiple scanning of the library symbol table so only one pass on the library itself is needed. This makes it unnecessary to order archives in any particular order. For more information concerning the mechanics of **ld**, consult the *UNIX System V Programmer Guide* and **ar(4)** in the *UNIX Programmer Reference Manual*.

OPTIONS

The following options are recognized by **ld**:

- e *epsym* Sets the default entry point of the output object file to the symbol *epsym*.
- f *fill* Sets the two byte value *fill* to be the default fill pattern for holes within output and initialized *bss* sections.
- lx Search the archive library *libx.a*, for *x* up to seven characters. Since libraries are searched in the order in which they appear in the command line, the placement of this switch is significant. The default areas for libraries

ld

are **/lib** and **/usr/lib**.

- m** Lists a map of input/output sections to standard output.
- o *outfile*** Places the resulting object file in *outfile* instead of *a.out*, the default.
- r** Saves relocation entries in the output object file. These entries must be saved if the output object file is to be used as an input file on a subsequent invocation of **ld**.
- s** Suppresses line number entries and symbol table information in the output object file.
- t** Suppresses warnings about multiply-defined symbols that are of unequal size.
- u *symname*** Enters *symname* as an undefined symbol in the symbol table. This can be used when loading entirely from libraries since an unresolved reference is needed to force the first routine to load.
- x** Discard local symbols in the output symbol table and enter external and static symbols only. This is a space saving measure.
- L *dir*** Changes the search for *libx.a* to *dir* before searching **/lib** or **/usr/lib** when preceding a **-l** option on the command line.
- M** Outputs a message for each multiply-define external definition. Note that extraneous output is produced if the objects being loaded include debugging option. (If the **-g** option is used with **cc**.)
- N** Puts the data section immediately following the text in the output file.
- V** Outputs the version of **ld**.

VS num Uses *num* as a decimal version stamp identifying the resulting object file. This stamp is stored in an option header.

EXAMPLES

The following merges *smallfile* and *another_smallfile* into the object file *bigfile*.

```
$ ld -o bigfile smallfile another_smallfile
```

DIAGNOSTICS

Self-explanatory

ALSO SEE

cc compiles C programs

NOTES

Input directives and options should insure the following properties for programs:

- C defines a zero pointer as null. Zero pointers cannot point to any object. Consequently, users must not place data in virtual address zero in the data space.
- The C compiler driver command **cc** automatically directs **ld** to link a startup routine with user programs. This routine adds a call to **exit()** at the end of the main program.

USES

nothing

link

link binds object modules

FORMAT

link [options] file1..n

DESCRIPTION

link binds together several object modules into one larger object module and resolves external references between the bound modules. If there are no errors, the resulting module should be executable by the system as a command.

link can be used to "pre-link" several object modules into a single one, which at a later time can be further linked.

link searches a library cross-reference file that describes the libraries of modules. It loops while searching until all possible externals, defined in the library and referenced by the module, are resolved (including those references added as a result of new modules).

OPTIONS

tbase=# Specifies the base address for the text section of the program. This is only applicable to stand-alone programs.

dbase=# Specifies the base address for the data section of the program.

sbase=# Specifies the base (low) address for the stack area. As the 68000 stack works down, the initial stack pointer when the program runs will be *sbase+ssize-delta* (where delta depends on what arguments and environment variables are passed). Normally *sbase* is not specified, permitting the operating system to choose a location (typically a whole segment, so it can grow). Putting a stack in the data segment is useful for saving a segment register, and possibly reducing total memory used. However, the stack cannot grow. If it is at the base of a segment, then the MMU (memory management unit) prevents overwriting good data if

link

the process overflows the stack.

l=libfile Causes libfile to be searched as a library file. The library files are searched in the order they were listed.

-l Saves local symbol table entries in the output file instead of discarding them.

-ns Causes the object file to be linked "non-sharable", with data immediately following the instructions. The resulting file may not be executable in user space on all UNOS systems.

This object format is appropriate for stand-alone programs, including the operating system, and is also the format typically used for modules intended for further processing by the linker (i.e., output from the assembler).

o=name Places the output in the specified file instead of the default **link.out**.

-r? Saves relocation information in the output file for further linking. Generates a file in the same format as the *-ns* option except that the file can be relinked so as to form either a user program or a stand-alone program.

ssize=# Specifies the size of the stack (in bytes) required to run the program. The system default is 4096 bytes.

-syms Generates a list of defined symbols and their values. This is a debugging tool for the linker.

u=name Causes the specified name to be considered an unresolved external reference.

-lun0 Indicates to the operating system that this program, when executed, expects LU numbers to begin with 0 for stdin.

-no_lun0 may be used to force stdin to be 1. The linker will default to the value appropriate for the library distributed in the same release, as the value of stdin (etc.) is compiled into dozens of library routines.

- sbd* Causes the stack area to be reserved at the beginning of the data section (at *dbase*). Overrides the *sbase=#* option.
- sbt* Causes the stack area to be reserved at the beginning of the text section (at *textbase*). Overrides the *sbase=#* and *-sbd* options.
- uv* Is equivalent to "*tbase=0 dbase=0x400000*" and disables *-sbd* and *-sbt*. This establishes the "standard" object format for the UV/68 product line.

EXAMPLES

The following command line links the two modules */lib/C68rt.j* and *ls.j* using the library file */lib/unos.lib* and places the output in the file, *ls*.

```
$link o=ls /lib/C68rt.j ls.j l=/lib/unos.lib
```

DIAGNOSTICS

Diagnostics are given for:

- missing files
- bad object code headers
- incorrect relocation formats
- errors in creating temp files
- bad options
- too many library entries
- multiply defined names

ALSO SEE

asm translates assembly language files

cc compiles C programs

genlib generates library entries from object programs and archives

version displays creation date and time of object files

NOTES

If *-uv* is not specified, the linker chooses default *tbase* and *dbase* addresses and a default for *-sbd* and *-sbt* according to the way the linker was created. Thus, a linker that runs on a particular system will make object files suitable to run on that system.

link

To properly link a standard C program, you must have **/lib/C68rt.j** as the first module and include the standard library.
`$link /lib/C68rt.j cobject.j l=/lib/unos.lib`

NOTE: When **link** is used by the C compiler, **cc**, this step is done automatically.

Order matters among libraries, because the first occurrence found in the first listed library is used; however, order does not matter between files and libraries.

The entire pathname of libraries need not be specified. If a library argument is given as *l=LIB* (where *LIB* contains no / characters), the following library files will be sought:

- /lib/libLIB.lib
- /lib/LIB.lib
- /lib/LIB
- /usr/lib/libLIB.lib
- /usr/lib/LIB.lib
- /usr/lib/LIB
- ./libLIB.lib
- ./LIB.lib
- ./LIB

The linker provides default values for four symbols if they are referenced but not defined: **end**
the end of the whole program.

edata

the end of the data section.

etext

the end of the text section.

__LUBASE

0 if **-lun0** is specified, or 1 otherwise.

USES

- /tmp/LDTMP.OD.#**
- /tmp/LDTMP.TR.#**

link

/tmp/LDTMP.DR.#
/tmp/LDTMP.SY.#

list lists the contents of directories

FORMAT

list [options] [file1...filen]

DESCRIPTION

list lists the filenames and additional information about the files specified.

If you do not specify any files, the current directory is assumed. Otherwise, for each argument, **list** determines if it is the name of a directory. If it is a directory, **list** lists its contents (unless the *-dir* option was used). If a file was specified, only its name is printed.

Additional information can be requested by using the options described below.

OPTIONS

all All.

Shows all files, including files whose names begin with "." which are normally considered "hidden" files.

-dir Directory.

Lists the specified directory ENTRY as opposed to the files in it.

-fileno File number.

Displays the file number which can be used for debugging purposes or finding out links.

-long Long.

Shows the length of the file in bytes, the access permission, the number of links to the file, the owner, the modification date, and the name of the file. If the file is a directory, a D precedes the listing. If the file is a device file, a U or B (Unbuffered or Buffered) is displayed followed by the major/minor device numbers. If the file is an eventcount,

list

the word **EVENT**-- appears. If it is a system eventcount, the word **EVENT** is followed by the eventcount number. If the year the file was created or modified is different than the current year, it is included in the listing.

-nosort Lists files without sorting them.

-m Lists files in multiple columns.

Won't work with **-l**, **-f**, or **-p** options.

-p Pathname.

Shows full pathnames rather than just the entry name.

Only works if the file arguments are full pathnames.

-r Reverse.

Reverses order of the sort. When used in conjunction with **-time**, displays modification from newest to oldest, otherwise the sort is reverse alpha.

-time Modification or Creation Date.

Lists the files in order of modification, oldest to newest.

-group In **-long** format, show group rather than owner.

EXAMPLES

To list the files *x*, *y*, and *z*, assuming they all exist, type

```
$list z y z
```

```
x
```

```
y
```

```
z
```

To see all the *.c files in order of time, as determined by their modification date, type

```
$list -time *.c
```

```
foo.c
```

```
cal.c
```

```
dd.c
```

```
al.c]
```

To see all the information about the file *work*, type

list

```
$list -l -g work
```

```
365 rw. rw. rw. (1) system   Aug 20 13:40:06 work
 1      2      3      4      5      6
```

This information, as numbered above, means:

1. size in bytes
2. read-write-execute permission owner others
3. number of links (names)
4. owner
5. modification date
6. filename

DIAGNOSTICS

If all of the files specified do not exist

File(s) do not exist.

appears, otherwise only the existing files are displayed.

ALSO SEE

addname gives a file another name

chown changes owner of files

delete deletes files

makedev creates a device file system entry

makedir creates directories

NOTES

You must have read permission to a directory in order to list its contents.

If your system administrator has done **addname 1 list**, the **l** command is the same as **list -l**.

USES

nothing

log

log maintains logs of program or system changes

FORMAT

* log [file1...filen]

DESCRIPTION

log reads its standard input to EOF, and prepends it to the file(s) given in the argument list with a header line stating author and date. If no filename is given, the default *Logfile* is used. For long log entries, use **ted** to write the information and then redirect it into the *Logfile*.

log is intended to provide a live on-line record of changes, such as variations in disk usage or a chronicle of the evolution of a project.

OPTIONS

none

EXAMPLES

A sample log entry:

```
$log /system.log
```

Today we deleted all files in /usr/games

DIAGNOSTICS

none

ALSO SEE

nothing

NOTES

none

log

USES

The *Logfile* in the current directory if no file is specified.

logger

logger collects timing information

FORMAT

logger [-debug] [logfile]

DESCRIPTION

logger collects process timing information from the system and writes it to the *logfile*. If no file is specified, it writes to standard out.

logger is an internal command. It is intended to be used only by the **profile** command; the interface is subject to change.

OPTIONS

-debug Prints information on each buffer read.

EXAMPLES

To turn on **logger**, set size in *conf* of histogram to
of seconds * 50 * 4
The entries per second is 50. The entry size is 4.

Example: 2000 for 10 second window.

DIAGNOSTICS

none

ALSO SEE

hist constructs a histogram of function usage
profile produces a function histogram

NOTES

none

logger

USES

nothing

login

login verifies user ID and password, and starts top-level shell

FORMAT

/sys/login [ttyname]

DESCRIPTION

login verifies the user id and starts the top-level shell.

When the startup program (**/etc/start**) enters multi-user mode, it starts a process for each line in the file **/etc/ttys** which is enabled (by the word "on" in the first field). Here is a typical line from **/etc/ttys**:

```
on:0:/dev/tty0:1/0:h19:speed=9600::System Console
```

login initializes the port to which it is connected by setting DTR low for a while (if the driver supports DTR) to force a modem to hang up, and sets the ttymodes back to a default state (specified by the sixth field in the line in **/etc/ttys**).

login then enters a loop in which it prompts **Name:** and waits for the user to type his name. It then prompts **Password:**, disables echoing, and waits for the user to type his password. If the name/password pair is not in the file **/etc/passwd**, **login** repeats this loop with a message **Login incorrect**. (Names are registered by the system administrator. Passwords are initially blank (just type <cr>) and can be changed with the **cpw** command.) Here is a typical **/etc/passwd** line:

```
admin:zwe7qz.2:0:3:::/bin/command
```

Once **login** has a valid user ID, it updates the file **/tmp/logins** to show who is logged in, using the second field of the line from **/etc/ttys** as a unique index into **/tmp/logins** so that file can be updated by multiple processes at the same time. **login** initializes the user's environment with the following:

```
LOGNAME=<user name, same as first field in /etc/passwd line>  
PATH=:/bin:/usr/bin
```

login

```
HOME=<sixth field in /etc/passwd line>
TERM=<fifth field in /etc/ttys line>
SHELL=<seventh field in /etc/passwd line>
MAIL=/usr/mail/<user name>
```

login sets the working directory, user ID, group ID, and ownership and modes of the tty. Finally it prints the file **/etc/motd**, prints **You have email.** if **/etc/mailboxes/<user name>.mail** exists, and executes the command processor (the same program named in the **SHELL=** variable).

The shell runs in the same process **login** used, which is a child of **/etc/start**. When the user logs out, he causes his shell to exit. **/etc/start** notices a child exiting, figures out which port it belongs to from the PID, and updates the file **/tmp/logins** to reflect the fact that no one is now logged in on that port. **/etc/start** creates another process running **login** for the port if it is still enabled (by the first field of the line in **/etc/ttys**).

login can also be used as a command. Omitting the *ttyname* argument to **login** causes it to skip some of the steps that are only appropriate for the system's use of **login**, such as resetting the ttymodes. Also, in this case, **login** runs the command processor in a child process so that it can put things back the way they were when the command processor exits.

OPTIONS

None

EXAMPLES

```
$who am i
rfm
$/sys/login
Name: jeff
Password:
$who am i
jeff
$ <EOF>
```

login

```
$who am i  
rfm  
$
```

DIAGNOSTICS

If **ttname** is specified but can't be opened, or can't be made the control terminal (source of ^C signals), **login** prints a message and sleeps forever. (Exiting in this case is likely to cause a loop.)

ALSO SEE

setuser changes the current user and group ID

Universe System Installation Manual for description of **/etc/ttys** and **etc/passwd**

NOTES

none

USES

/etc/ttys
/etc/passwd
/tmp/logins
/dev/tty#

lorder

lorder finds ordering relation for an object library

FORMAT

lorder file ...

DESCRIPTION

lorder cross references library archive files to produce a list that can order a library suitable for one-pass access by **ld**. Each entry in the list comprises two object file names

1. the first file is the name of the file under consideration
2. the second file is a file **lorder** has determined to contain external definitions referenced by the first.

This list can be further processed to find an ordering of a library suitable for one-pass access by **ld**. Although one-pass access is not generally necessary, cases exist when this type of access is slightly more efficient than the usual multi-pass invocation.

OPTIONS

none

EXAMPLES

The following builds a new library from existing .o files:

```
$ ar cr library 'lorder *.o | tsort'
```

DIAGNOSTICS

none

ALSO SEE

ar archive and library manager for portable archives

ld links editor for common object files

lorder

NOTES

Object files whose names do not end with .o, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

USES

nothing

lpd runs the line printer from a queue

FORMAT

`/sys/lpd [tail=#] [printer]`

DESCRIPTION

lpd dequeues line printer requests. The line printer daemon process is created when the system goes from single to multiuser. The `/etc/startup` file creates the daemon. For each printer on your system, you need a separate daemon.

OPTIONS

`tail=#,t=#` sets the number (#) of form feeds (blank pages) printed after each print job finishes. The default is 1.

EXAMPLES

To create a line printer daemon for the centronics printer, add this line to the startup file:

```
/sys/lpd centronics %/dev/null >/dev/null &
```

All errors and all standard out messages are directed to `/dev/null` (no where). The command is always run as a background process.

DIAGNOSTICS

none

ALSO SEE

lpr queues printing requests

UNOS System Concept Manual

lpd

NOTES

After each print job, and before the form feeds produced by the *tail* option, **lpd** will issue a carriage return. The carriage return clears the print buffer on centronics printers. If your printer issues a line feed when a carriage return is received, and if *tail=0*, the extra carriage return may misalign your paper.

USES

/tmp/_qtemps/{printer}.*

queues

/tmp/_queues/ambles/{printer}.queue.{style}.{pre!post}

preamble/postamble files

lpmaint maintains printer queue

FORMAT

lpmaint [**printer**][**options**]

DESCRIPTION

lpmaint provides two functions: it changes the status of the entries in the printer queue and it fixes an inconsistent printer daemon. The requester and system administrator can change the status of a printer request, but only the system administrator can "fix" the daemon.

OPTIONS

All options can be abbreviated to their first letter.

- kill #1...n** Kills printing request(s) as specified by the request number. Depending on the timing of the kill command, the request is either removed from the queue and not printed, or stopped while printing.
 - stop #1...n** Stops or suspends printing request(s). If the request is waiting to be printed, it never prints. If the request is printing, it is stopped. In all cases the status is changed to "stopped" on the queue.
 - resume #1...n** Resumes stopped request(s). The status is changed to waiting, and the request is printed in turn from the queue.
 - fix** Fixes the printer daemon. Checks the printer queue and returns it to a consistent state. Used only when starting up the system (usually written into the /etc/startup command file).
- Do not use when the printer daemon is running.**

EXAMPLES

To stop requests 2 and 4 and kill requests 1 and 5, type
\$lpmaint -stop 2 4 -kill 1 5

lpmaint

DIAGNOSTICS

none

ALSO SEE

lpq displays status of printer queue

lpr queues printing requests

UNOS System Concepts Manual

NOTES

none

USES

/etc/printers For the location of the printer queue.

/tmp/__queues/printer.kill The kill eventcount.

/tmp/__queues/printer.info The name and packet number.

lpq displays status of printer queue

FORMAT

lpq [*printer*]

DESCRIPTION

lpq displays the entries in the printer queue. If no printer is specified the default printer is assumed.

Information about the printer queue is displayed in fields:

req Request number.

This is the number the request was given when it was enqueued. The number can be used by **lpmaint** to change the status of the queue.

pri Priority.

Can be either LO (low priority), HI (high priority) or blank (normal priority).

state Current state.

Can be either working (being printed), waiting (waiting to be printed), or stopped (suspended by **lpmaint**).

user Name of the print requester.

file Pathname of the file requested.

alias Name used to notify the requester.

Number of copies requested.

D Whether to delete the file after printing.

N Whether to notify the requester of file printing.

OPTIONS

none

lpq

EXAMPLES

\$lpq

default queue:

req	pri	state	user	file	alias	#	D	N
10	HI	working	dianne	/tmp/lpr0.128	docs	5	Y	Y
3		stopped	jeff	/src/cmds/lpq.c	lpq	1		Y
12		waiting	dano	/usr/dano/dved	dved	1		
11	LOW	waiting	marjie	/usr/marjie/format	fmtq	1		Y

DIAGNOSTICS

none

ALSO SEE

lpmaint maintains printer queue

lpr queues printing requests

UNOS System Concepts Manual

NOTES

none

USES

/etc/printers For the location of the printer queue.

/tmp/_queues/printer.kill The kill eventcount.

/tmp/_queues/printer.info The name and packet number.

lpr queues printing requests

FORMAT

lpr [options] file...filen

DESCRIPTION

lpr enqueues its argument files for printing. If there are no arguments, it reads from standard input into a temporary file and enqueues this temporary file for printing.

The text sent to the printer is not processed in any way.

OPTIONS

All options can be abbreviated by their first character.

alias=name Uses name instead of filename when notifying the user.

count=#,-# Specifies the number of copies to print.

-delete Deletes the file when the printing is done.

-high Prints the request at high priority, before normal and low priority requests.

-low Prints the request at low priority, always printed last.

-notify Notifies the user when the printing is done.

pbl=style For printers with switchable modes (i.e., printer pitch), this option enables **lpd** to output sequences to enter and exit the desired style. For instance, a laser printer which has a DIABLO simulation mode could be used with the following command line:

```
$lpr pbl=diablo filename
```

printer=name Prints the file on specified printer.

lpr

-raw Ensures that the print request is performed with **no** character translations (e.g., newline to CRLF, tab expansion).

EXAMPLES

To print the files *temp*, *.globals*, and *Makefile*, type
`$lpr temp .globals Makefile`

The following command line prints two copies of the file *expenses*, and upon completion notifies the user, with the alias name *ex_report*.

```
$prfmt expenses | lpr -notify alias=ex_report -2 -high
```

The following command line prints a temporary file at low priority on the spinwriter and upon completion deletes the file.

```
$lpr temp printer=fancy -low -delete]
```

The next command line is a macro definition that prints the specified files on the spinwriter, and uses the first file name to notify the requester.

```
 $#b printf do "prfmt $r1 | lpr p=fancy -n alias=$1"
```

To use the macro, type

```
$printf file1 file2
```

From admin 15:16:49: Printing of file1 is complete.

DIAGNOSTICS

none

ALSO SEE

format formats text files for printing

lpmaint maintains printer queue

lpq displays status of printer queue

prfmt paginates files for printing

UNOS System Concepts Manual

NOTES

To find out the names of the printers on the system, check the file */etc/printers*. The printer queue is assumed to be

lpr

/tmp/_queues/line_printer_queue unless otherwise specified.

To learn which printers have running daemons, use the **ps -a** command and look for the file, **/sys/lpd printer_name**.

If you have UniverseNet, you can use **lpr** over the network. See the *UniverseNet Networking Administrator's Guide* for details.

USES

/tmp/lpr* For temporary files.

/etc/printers To find the location of the printer queue.

makedev

makedev creates a device file system entry

FORMAT

makedev name [u or b] major minor

DESCRIPTION

makedev creates a device file system entry. The first argument is the name of the entry. The second is either *u* for unbuffered (direct access to the device driver) or *b* for buffered (access through the system buffers). The last two numbers specify the major and the minor device types (e.g., unit, drive, or line number). These numbers are assumed to be decimal.

You must have admin privileges to use the **makedev** command.

OPTIONS

u Unbuffered

b Buffered

EXAMPLES

To create the null device entry, type

```
$makedev /dev/null u 0 0
```

To create the floppy drive 0 entry, type

```
$makedev /dev/cfd0 b 4 6
```

DIAGNOSTICS

none

ALSO SEE

nothing

makedev

NOTES

The assignment of major and minor device numbers is system specific. See *UNOS System Concepts Manual*.

Device files can exist in any directory, by convention they are in **/dev**. Although a device can have multiple filenames, it is not recommended.

USES

nothing

makedir

makedir creates directories

FORMAT

makedir directory1...directoryn

DESCRIPTION

makedir creates new directories with the given pathnames. All higher level directories in each pathname must already exist.

OPTIONS

none

EXAMPLES

To create a subdirectory called *jdoc* in user john's directory, type
\$makedir /usr/john/jdoc

DIAGNOSTICS

**makedir: <system error message>. Cannot make directory
'<name>'.**

ALSO SEE

nothing

NOTES

It is an error to try to create a directory using a name that already exists.

makedir will try to create each directory specified even though some of the attempts will fail. For example the command line
\$makedir a b c will create directory 'c' even though the creation of 'a' or 'b' may fail.

makedir

USES

nothing

makeec

makeec creates eventcount file system entries

FORMAT

makeec [s=#] file1...filen

DESCRIPTION

makeec makes a file system eventcount, which can then be used by programs.

OPTIONS

sys=#, s=# Installs system eventcounts (e.g., **/dev/clock**).

EXAMPLES

To make the system clock entry, type
\$makeec s=1 /dev/clock

To create a normal user eventcount, type
\$makeec /usr/jeff/sync.ec

DIAGNOSTICS

makeec: <system error message>.
Cannot make ec'<name>'.

ALSO SEE

advance increments eventcounts
list lists the contents of directories
printec prints eventcount values

NOTES

Making an eventcount with the same name as an existing file removes the existing file entry and its value and replaces it with the new zeroed eventcount entry. If the eventcount already exists, its value is left unchanged.

makeec

Currently defined system eventcount numbers are:

sys=1 System clock

This is advanced at fixed intervals (1/10 second).

sys=2 Child exits

This is advanced whenever a child process exits or suspends.

The eventcount is per-process and references always get the referencing process' eventcount.

The **sys=#** parameter applies to **all** the eventcounts created.

makeec exits if any creation fails and will not continue making the rest of the eventcounts.

Only the **/dev/clock** system eventcount is available to users.

Eventcounts can be deleted with the **delete** command.

USES

nothing

makefs initializes a file system

FORMAT

makefs device size [*files*]

DESCRIPTION

makefs creates a UNOS file system on the specified device by writing at the beginning of the device the following information: the file system header, file descriptors, and the bit maps.

The *size* is specified in blocks and is the total number of blocks on the specified device. The optional argument *files* is the maximum number of files expected on the file system. If the number is omitted, **makefs** uses 9/64 of the number of blocks.

makefs allocates the first 6 blocks at the beginning of the file system to account for volume ID, test block, bad block list and three boot blocks. The test block is changed to indicate the file system is not mounted. It also rounds up the number of file descriptors to fill out a block. Finally, to prevent earlier version file system headers from being found, **makefs** erases any block found to contain a file system header that would be physically lower in block address than the one it is currently writing.

OPTIONS

fsblk=*<number>* sets the block that will contain the file system header. The default block is 9.

EXAMPLES

This example shows how to make a file system on a floppy with a total of 160 files.

Insert a formatted write enabled floppy into the drive.

At the console type the following commands:

```
$fmtdisk /dev/rcfs0
```

```
Format /dev/rcfs0: are you sure? yes
```

makefs

```
$makefs /dev/rcfs0 494 160  
making freemap  
making fdmap with 160 fds  
making new fd's  
468 blocks available, writing header
```

Remove floppy from the drive.

DIAGNOSTICS

none

ALSO SEE

fmtdisk formats a hard or floppy disk
fscheck checks and fixes file and directory structures
mount mounts a file system
unmount unmounts mounted file systems

NOTES

The total number of blocks on the device is given to the command. **makefs** determines the total number of usable blocks after the allocation bit maps and file descriptors are created. Once a file system is created, the total number of files is fixed.

makefs works faster when you specify the "raw" device.

Do not exceed 32767 file descriptors in any particular file system. For file systems over roughly 200000 blocks, **makefs** chooses 32767 file descriptors by default. 32767 does not completely fill the last block of file descriptors; this is the only case in which **makefs** does not round the number up to a block boundary.

Maked the default permission of the created root directory rwxrwxrwx rather than rwx ... rwx.

USES

nothing

match

match searches for matching text strings in files

FORMAT

match [-s][-n][-not] pattern [file1...fileN]

DESCRIPTION

match searches the named files or standard input (if no filenames are given) for the occurrences of the given pattern on each line. The program accepts literal characters or special pattern matching characters. All lines that match the pattern are sent to standard output. You can only specify one pattern string for each **match**, however, you can construct an arbitrarily complex string. When you do not specify a file, **match** can be used as a filter to display desired lines.

OPTIONS

- s Suppresses the printing of filenames.
- n Prints the file line number (with respect to the input file) of each output line.
- not Prints all lines that do not match.

EXAMPLES

The following command line displays on standard output all lines containing **printf** in all the *.c files in the current directory.

```
$match printf *.c
```

The next command line displays only the lines of the directory listing that contain the word **admin**.

```
$list -l | match admin
```

DIAGNOSTICS

match: File does not exist. Can not open 'file name'.

match

ALSO SEE

change substitutes text strings in files

NOTES

See *User's Guide to UNOS* for the pattern matching conventions.

Even if a match occurs more than once per line, the line is output only once.

Quote special pattern matching characters to prevent them from being expanded by the Command Interpreter.

There are limits to the sizes of lines and patterns: maximum line length is 2048, maximum pattern length is 256.

USES

nothing

message

message allows or prevents writes to terminal

FORMAT

message [on/off]

DESCRIPTION

message inspects, sets or clears the permission bits which determine whether other users can write to the terminal.

To find out the current state:

```
$message
```

To enable messages:

```
$message on
```

To disable messages:

```
$message off
```

OPTIONS

none

EXAMPLES

To enable the terminal so that it can now receive messages, type
\$message on

DIAGNOSTICS

none

ALSO SEE

write sends a message to specified users

wall sends a message to all users currently logged on

who displays names of users currently logged on

message

NOTES

The access permission on your tty entry controls message reception. When you log on to the system, you are given ownership of your tty. Thus you can change the access modes through the **message** command. The default is reception **on**.

USES

/etc/ttys To determine current terminal.

mnetr receives a file via serial tty ports

FORMAT

mnetr *file_name* [-lock] /dev/ttyN </dev/ttyN>

DESCRIPTION

mnetr receives data from stdin and writes it to *file_name* and sends acknowledgements to stdout. The file is received as a series of 36-byte packets and it assumes that the utility **mnett** is operating on the opposite Charles River machine.

mnetr will automatically terminate upon completion of received data or when one of the following error conditions occur:

1. A failure to receive an initial signal from **mnett** on the opposite machine within 30 seconds of startup.
2. Ten consecutive check-sum errors on the same packet.
3. Failure to receive a complete packet from **mnett** within 30 seconds of acknowledgement of the previous packet
4. An abort message sent from **mnett** (usually due to a nonexistent or protected file).

OPTIONS

-lock filelocks stdin and stdout

EXAMPLES

In the following example, communication is through the serial port tty3. Since **mnetr** is expecting to receive data packets from stdin and to send acknowledgements to stdout, redirect stdin and stdout to /dev/tty3. **mnetr** will place all valid data in the file foo.c.

```
$mnetr foo.c >/dev/tty3 </dev/tty3
```

mnetr

ALSO SEE

frece allows a remote Charles River machine to receive a file from a host

ftrans allows a remote Charles River machine to send a file to a host

mnett transmits a file via serial tty ports

mnetr receives a file via serial tty ports

term asynchronous terminal emulator

NOTES

login must not be running on the tty port used by both the transmitting and receiving machines. The baud rate for the tty port used by both machines must be the same.

USES

nothing

mnettt transmits a file via serial tty ports

FORMAT

mnettt *file_name* [-lock] /dev/ttyN </dev/ttyN>

DESCRIPTION

mnettt transmits *file_name* to stdout and receives acknowledgements from stdin. The file is transmitted as a series of 36-byte packets and it is assumed that the utility **mnettr** is operating on the opposite CRDS machine.

mnettt will automatically terminate upon completion of transmission or when one of the following error conditions occur:

1. A failure to receive an initial acknowledgement signal from **mnettr** on the opposite CRDS machine within 30 seconds of startup.
2. Ten consecutive checksum errors on the same packet.
3. Failure to get an acknowledge from **mnettr** within 30 seconds after transmission of a packet.
4. An abort message sent by **mnettr** (usually due to an attempt to write over a protected file).

OPTIONS

-lock filelocks stdin and stdout

EXAMPLES

In the following example, communication is through the serial port tty3. Since **mnettt** is expecting to send data packets from stdin and to receive acknowledgements to stdout, redirect stdin and stdout to /dev/tty3. **mnettt** will send all valid data to the file foo.c.

```
$mnettt foo.c >/dev/tty3 </dev/tty3
```

mnett

DIAGNOSTICS

Self-explanatory errors are reported to stderr

ALSO SEE

frexv allows a remote Charles River machine to receive a file from a host

ftrans allows a remote Charles River machine to send a file to a host

mnetr receives a file via serial tty ports

term asynchronous terminal emulator

NOTES

login must not be running on the tty port used by both the transmitting and receiving machines. The baud rate for the tty port used by both machines must be the same.

USES

nothing

mount

mount mounts a file system

FORMAT

mount [device directory [-r]]

DESCRIPTION

mount notifies the system that a removable file system appears on *devicefile*. Specify the root name of the device to mount as *directory*. If no arguments are provided with the command, **mount** prints a current table of mounted devices.

OPTIONS

-r Indicates the file is mounted as read-only. Mount physically write-protected and magnetic tape file systems this way or errors occur when access times are updated -- regardless of whether or not any explicit write is attempted.

EXAMPLES

To mount the directory *floppy_archive_10* on *floppydrive0*:
\$mount /dev/floppydrive0 /floppy_archive_10

DIAGNOSTICS

mount displays a warning if the file system you want to mount is mounted.

SEE ALSO

umount unmounts a file system

NOTES

none

USES

nothing

move renames files or moves them across file systems

FORMAT

move [options] [from_files] [to_file!directory]

DESCRIPTION

move renames a file by changing the *from_file* name to the *to_file* name. The *to_file* is overwritten, if it existed, and the *from_file* is deleted.

If the **move** crosses file system boundaries, then it is implemented as if the following sequence of commands had been given

```
$copy file1 file2; delete file1
```

Should, however, the destination file system have no space, no delete occurs, although the destination entry might be partially created.

If more than one file is specified on the command line and the last one is a directory, all previously specified files are moved to that directory. This is true for all types of files -- directories, devices, eventcounts, named pipes, and source files.

If the last file on the command line is *not* a directory, specifying more than two files causes an error. The first file is the source and second the target.

- If the second file exists, a copy of the first file is written into the second, preserving all links to the second file.
- If the second does not exist, it is created and a copy of the first file is written into the second.

Unless you specify the *-interactive* option with less than two file names, the command line is rejected as syntactically invalid.

You must have write permission to the destination file, if it already exists. The *-force* option (described below) lets you copy to a file even if you don't have write permission. (But you must have write permission to the directory.)

move

If the *from__file* is a directory, **move** creates not only the target directory, but recursively copies all subfiles.

When the command is done, you'll see an exit message that indicates what was accomplished:

- If no files are moved, you'll see the message:
No files moved.
- If one file was moved, you'll see the message:
One file moved.
- If two or more files were moved, you'll see the message:
2 files moved
(or 3..4..5...etc)
- If one file was not moved, you'll see the message:
One file not moved.
- If two or more not moved, you'll see the message:
2 files not moved.
- If some files were moved and others were not, you'll get two messages telling how many were moved and how many were not.

Use the *-silent* flag to avoid printing any messages.

OPTIONS

All options can be abbreviated to the first letter, except *-setowner* and *-settime*.

- force* Use this option to move to files for which you do not have write permission. You must have write permission to the directory.
- help* Prints, to standard error, a few lines listing all the command options **move** accepts.
- interactive* Prompts the user for more files after moving any files given on the command line. You are first prompted for the file to move from, and are then prompted for the file to move to. Type a ^D or a press the RETURN key on a blank line to end the command.
- merge* No overwrite. Prevents files with duplicate names from being overwritten.

move

- query* Asks whether a file should be moved. A response of either **yes** or **y** moves the file; any other response is interpreted as negative.
- recursive* If the **from_file** is a directory, creates not only the target directory but recursively moves all subfiles. This option is always on by default.
- silent* Prints no messages to standard out. No exit message is printed.
- setowner* Attempts to move the original user id.
- settime* Duplicates the modification time of the source file for the destination file.
- verbose* Lists files that have been moved.

EXAMPLES

\$move alpha beta

Changes the name of the file from *alpha* to *beta*.

\$move /usr/jeff/fun_and_games/new_game /games

Moves the file *new_game* to the directory */games*.

DIAGNOSTICS

Could not delete directory <filename>

When moving a directory, it could not delete the old directory.

Last file does not exist <filename>

If more than two files are specified on the command line, the last file must exist.

Last file is not a directory <filename>

If more than two files are specified on the command line, the last file must also be a directory.

Could not open <filename>

File to be moved could not be opened.

move

Write access denied to <filename>

The destination file does not have write access and the **-force** option was not used.

Could not delete <filename>

Could not delete the old file, before creating the file to move into. If you are moving into an existing file of a dissimilar type, an attempt is made to delete the "to" file.

Can not make directory <filename>

Could not create directory after deleting the old file.

Cannot read from <filename>

Could not read from the file to be moved.

Cannot move <filename> to itself <filename>

You cannot move a file to itself.

Overwrite not allowed of <filename> with <filename>

If **-merge** option is used **move** will not overwrite a file with the same name.

Cannot create named pipe <pipename>

If moving a pipe, could not create a new pipe.

Cannot open <filename> for writing

Could not open the destination file to allow write access.

Could not make event <eventname>

If moving an eventcount, could not create a new eventcount.

Could not make device <devicename>

If moving a device, could not create a new device.

Unable to set access modes of <filename>

Could not set the access modes of the destination file to be that of the **from_file**. The file has already been moved.

Unable to set owner of <filename>

-setowner option was specified in the command line but was not able to change the owner. The file has already been moved.

move

Could not set modified time of <filename>

-settime option was specified in the command line but was not able to set the time as requested. The file has already been moved.

Unable to open newly created file <filename> Destination file created but inaccessible.

ALSO SEE

addname gives a file another name

copy duplicates files

delete deletes files

NOTES

If the new name already exists in the file system, **move** first tries to delete it.

USES

nothing

namelist

namelist displays symbol table of the old *rel* object files

FORMAT

namelist [-b][-d][-g][-n][-nosort][-r][-t][-u][-v]
file..n

DESCRIPTION

namelist prints out in sorted order the following information: symbols, types, and values (in hex) for selected object files. The default is all symbols in ascending alphabetical order.

OPTIONS

- b bss symbols.
- d data symbols.
- g global symbols.
- n sort by numeric symbol value.
- nosort no sort, displays symbols in order of occurrence.
- r reverse order of sort (to descending).
- t text symbols.
- u undefined symbols.
- v verbose, precedes each symbol with the associated file.

EXAMPLES

To find out all external references in a file, type
`$namelist -g -u file.j`

namelist

The next command line finds the file in a group of files that refers to a certain external symbol and prints the file name and its references.

```
$namelist -v *.j | match xx
```

DIAGNOSTICS

Bad header in 'file'.

The file is not an object file.

Read error of unexpected EOF.

The file is either a partial file or not an object file.

No relocation in file 'file'.

The file has been stripped of symbol table information.

ALSO SEE

cc compiles C programs

link binds object modules

strip strips symbol and line number information from a common object file

NOTES

namelist runs considerably faster if you use only one option.

namelist sorts on the second key. For example, if you specify a numerical sort, you would get the following output:

value	name
1	A
1	B
1	C
2	A
2	B

where items that have the same numerical value are then sorted alphabetically.

namelist

On the other hand, if you specified an alphabetical sort the output would be:

value	name
1	A
2	A
1	B
2	B
1	C

Use the **nm** command with COFF files.

USES

nothing

newgrp

newgrp enters a new group

FORMAT

newgrp group

DESCRIPTION

newgrp spawns a new command interpreter that executes with the group identification specified on the command line. The command interpreter chosen is the one specified by the SHELL environment variable.

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

OPTIONS

none

EXAMPLES

```
$list -l -d -g /src
```

```
D 160 rwx rwx ... (1) other Jul 6 14:55:19
/usr/evan/tmp
```

```
$list -l /src
```

```
list: Access permission denied. Error opening directory '/usr/evan/tmp'.
```

```
$newgrp other
```

```
Password:
```

newgrp

```
$list -l /src
```

```
/src:
```

D	1056	rwX	...	rwX	(1)	admin	May	9	21:08:21	boots
D	5216	rwX	...	rwX	(1)	admin	Jul	5	11:35:12	cmd
D	576	rwX	...	rwX	(1)	admin	Jun	25	19:21:38	games
D	2176	rwX	...	rwX	(1)	admin	Jul	6	12:39:38	header
D	1024	rwX	...	rwX	(1)	wjr	Jun	7	19:22:29	iop
D	288	rwX	...	rwX	(1)	admin	Mar	13	19:36:55	lang
D	5952	rwX	...	rwX	(1)	admin	Jun	26	15:07:10	lib
D	1216	rwX	...	rwX	(1)	admin	Jun	16	02:11:56	sys

DIAGNOSTICS

can't exec shell

When the proper command interpreter cannot be run.

Can't set group ID

A problem with the modes of the **newgrp** program.

Say, who are you!?

When the current user cannot be found in **/etc/passwd**.

Sorry.

When an incorrect password is given.

no group

The group specified is not mentioned in **/etc/group**.

ALSO SEE

chgrp changes the group associated with a file

setuser changes the current user and group ID

su changes the current user id

NOTES

If you want to use **newgrp** in the C shell, change the name of **/bin/newgrp**. The C shell has its own **newgrp** command. If you try to invoke the UNOS **newgrp**, while using the C shell, the process suspends.

newgrp

Group passwords are difficult to enter in **/etc/group**, and they represent a serious security risk in any case.

USES

/etc/passwd To identify the user by name

/etc/group To look up the name of the group

nice executes a command at a given priority

FORMAT

nice [*ceil*=#][*floor*=#] [-*lock*] *command line*

DESCRIPTION

nice sets the priority and/or locks the process resident. Since the command does not go through the Command Interpreter, only a simple command line can be used. If a complicated command line or abbreviation is specified, use

nice *command* -c "*abbreviation*"

Be sure to quote quote the abbreviation if it has several arguments, to ensure correct interpretation.

OPTIONS

ceil=# Sets the ceiling priority of the process.

If the ceiling is not specified, **nice** lowers it by 40. Only the system administrator (or a user who has used **setuser** to change to admin) can raise priority.

floor=# Sets the floor priority of the process.

The floor is set to 0 by default.

-*lock* Locks a process in memory.

Only the system administrator (or a user who has used **setuser** to change to admin) can lock processes.

EXAMPLES

In the following command line, the abbreviation **ff** is format
`$rl | lpr p=fancy -notify` and is expanded by **command**.
\$nice *command* -c "**ff** chap.*"

DIAGNOSTICS

none

nice

ALSO SEE

notify runs program in background and reports completion

setpri sets priority of a process

NOTES

This command can effect system performance drastically -- high priority processes can prevent other processes from ever running.

USES

nothing

nm displays the symbol table for a COFF file

FORMAT

nm [-efhnouv~~x~~VT] files

DESCRIPTION

nm displays the symbol table for the COFF files specified. The files specified may be relocatable or absolute common object files, or archives of relocatable or absolute common object files. **nm** prints the following information for each symbol:

Name The name of the symbol.

Value The name's value expressed as either an offset or an address, depending on its storage class.

Class The symbol's storage class.

Type The symbol's type and derived type. If the symbol is a structure or a union then the structure or union tag will be given following the type (struct-tag). If the symbol is an array, then the array dimensions will be given following the type (for example, *int[x][y]*). Note that in order for type information to be obtained, the object file must be compiled with -g option of **cc**.

Size The size of the data represented by the symbol measured in bytes. This information can only print if available and if the object file has been compiled with the -g option of the **cc** command.

Line The line number in the source where the symbol is defined -- if available. In order for this information to appear, the object file must be compiled with the -g option of the **cc** command.

Section The object file section which contains the symbol (*text*, *data*, or *bss*). This is reported only for static and external storage classes.

nm

OPTIONS

- e Print external and static symbols only.
- f Print redundant symbols (*.text*, *.data*, and *.bss*) which are usually suppressed.
- h Suppress output header data.
- n Sort external symbols by value.
- o Use octal printouts for value and size of symbols (the default is decimal).
- u Output undefined symbols only.
- v Sort external symbols by value.
- x Use hexadecimal printouts for value and size of symbols (the default is decimal).
- V Output the **nm** version to standard error.
- T Truncate symbol names listed to preserve the continuity of columnar output. This option keeps symbols of unordinary length from causing columns in the printout from becoming misaligned. If invocation of the -T option causes the truncation of a symbol name, it is denoted by an asterisk where the truncation occurs.

EXAMPLES

To list `/bin/nm`'s symbols:

```
shell> nm /bin/nm
```

Symbols from `/bin/nm`:

Name	Value	Class	Type	Size	Line	Section
nm	2	file				debug
/lib/c68rt0.o	3600	file				debug
start	0	static				.text
nm.o	3600	file				debug
.lfake	0	strtag				debug
_ptr	0	strmem	uchar*. 4			abs
_cnt	4	strmem	long. 4			abs
_base	8	strmem	uchar*. 4			abs
_flag	12	strmem	short. 2			abs
etc.						

DIAGNOSTICS

Self-explanatory

ALSO SEE

ar archive and library manager for portable archives
ld links editor for common object files

NOTES

When **nm** is printing all symbols, they must be printed in the order in which they appear in the symbol table so that scoping information can be preserved. Consequently, the `-v` and `-n` options should be used only in conjunction with the `-e` option.

USES

`/usr/tmp/nm?????`

nohup

nohup runs a command immune to hangups and quits

FORMAT

nohup command [arguments]

DESCRIPTION

nohup executes *command* with hangups and quits ignored. If you do not specify a redirection of the output, standard output and standard error are written to *nohup.out*. If you can't write to *nohup.out* in the current directory, output goes to **\$HOME/nohup.out**.

OPTIONS

none

EXAMPLE

To run a number of commands under the auspices of **nohup**, place those commands in a shell script. Then run **nohup** in the shell script.

```
nohup ksh file
```

DIAGNOSTICS

Self-explanatory

SEE ALSO

chmod sets or changes modes on files

nice executes a command at a given priority

NOTES

nohup command1; command2

nohup applies only to *command1*

nohup (command1; command2)
is syntactically incorrect.

nohup

Be careful of where standard error is redirected. The following command may put error messages on tape, making it unreadable:

```
nohup cpio -o <list >/dev/rmt/lm&
```

while

```
nohup cpio -o <list >/dev/rmt/lm 2>errors&
```

puts the error messages into file errors.

USES

nothing

notify

notify runs program in background and reports completion

FORMAT

notify [*ceil=#*][*floor=#*][*e=file*][*o=file*] program

DESCRIPTION

notify runs a program in the background and reports when it has completed. Unless otherwise directed, all standard output and error messages are placed in a special file. By default **notify** creates, using the process id of the backgrounded command, a file named *pid.out*.

When **notify** is executed, the process id of the command is printed on the console. **notify** waits for the command to complete and then prints out:

**command: <pid> completed due to <reason> returning
<number>**

the completed due to displays the reason for completion, whereas the **returning value** is the value set at the exit of the program. This message overwrites anything on the screen and is accompanied by a beep to call attention to its once-only nature. **notify** put the output of the program in the file, *pid.out*.

OPTIONS

ceil=# Sets the ceiling priority of the command. By default the ceiling of a backgrounded process is lowered by 20.

floor=# Sets the floor priority of the command which is by default 0.

e=filename Redirects standard error to the name file.

o= filename Redirects standard output and standard error (unless the **e=** option was specified) to the named file instead of the **pid.out** file.

notify

EXAMPLES

```
$notify o=makeout build testprog
223
```

```
.
.
.
```

build: 223 completed due to normal completion returning 0

```
$cat makeout
```

To look at the results of **build**.

DIAGNOSTICS

none

ALSO SEE

nice executes a command at a given priority

NOTES

notify ignores SIGHUP as well as SIGINT and SIGOUT.

USES

pid.out uses the process identification number for all standard error and standard output information generated by the program, when run in the background, unless the *o=* and *e=* options were specified.

p paginates files and displays non-printing characters

FORMAT

p [length=#][width=#][-silent][-raw] [file..n]

DESCRIPTION

p reads its argument files or, if no file names are given, standard input, and displays them on standard out. It paginates, stopping at the end of each screenful with the question **MORE?** The question can be answered in the following ways:

y,Y,<cr>,LINEFEED Yes.

The next screenful of text is displayed.

h,H Half.

The next half screen of text is displayed.

q,Q Quarter.

The next quarter screen of text is displayed.

l,L Line.

The next line of text is displayed.

1-9 Lines.

The next 1-9 lines are displayed.

n,N No.

No more of this file is displayed. Go on to the next file.

s,S Stop.

Exits the program.

other

Any other response causes the terminal to beep.

At the end of a file, **p** moves to the next file, and prompts the user for more.

P

p also interprets nonprintable characters, such as control characters, by printing ^ followed by a transformed version of the character, for example ^C. Note that spaces and tabs (which are defined at every eighth column) are treated normally. When lines exceed the specified width, they are displayed as wrapped around -- and continued on the next line of the screen.

OPTIONS

All options can be abbreviated by their first letter.

length=# Sets the number of lines on a page.
The default is 24.

width=# Sets the number of characters in a line.
The default is 79.

-raw Raw mode.
Control characters are not expanded.

-silent No prompts are given to the user.
Everything is printed. Use this option when you re-direct the output to a file.

EXAMPLES

To paginate the file **p.c** on the screen, type
@>p p.c

To print a "visible" copy of the current directory (which will display a lot of "garbage" characters), type
@>p . -silent | lpr

To paginate the output of **format** without filtering terminal control commands, type
@>format dev=heath chap.01 | p -raw

DIAGNOSTICS

none

ALSO SEE

cat concatenates files

prfmt paginates files for printing

NOTES

If you execute **p** without arguments and do not redirect its standard input, **p** will read from standard input. It will appear that the system is not responding. To exit this situation, type either **^C**, **^D**, or several **<cr>**'s until the **MORE?** prompt is displayed, then answer with **n** or **N**.

If the eighth bit of a character is "on" on your terminal, the tilde character **~** will appear before the character, i.e., **~C**. If the character has the eighth bit "on" and is also considered a control character, then the tilde will precede the escape control character sequence, i.e., **~^C**.

USES

nothing

package

package is a general purpose file dumping utility that writes files to a tape or disk in an **installation** (cpio) file format.

FORMAT

package [bs=buffersize] [after=date] [-s] [-q] [-c]
[-recover] [-help] [-nr] [-x] [-v] [-tape] [patterns]

DESCRIPTION

package writes a tape in UNIX cpio format, which the UNOS **installation** program reads. Use **package** to interchange files with SYSTEM V sites and other Charles River computers.

package also

- recovers files from filesystems with bad blocks
- reports any unreadable blocks
- backs up entire running systems by file modification date
- *streams* a 1/4-inch tape by using the *bs=* option to specify the amount of buffer memory to use.

In normal operation, **package** reads the files to output from the command line and recursively dumps the files to standard output. For example, **package** walks the directory tree. Walking the directory means to dump files on a partition, like */usr*.

Make sure the partition is mounted. Use **package** to dump an entire system at once. If no files are specified on the command line, **package** reads the standard input for the file names to dump. This is useful for setting up *take* files with the **list** command. If **package** comes to the end of file on an output medium, it requests a new volume to begin writing to.

OPTIONS

bs=buffersize Denotes buffer size. The default buffer size is 5120 bytes. The default buffer size accomodates disks, but it's small for tapes. Specify the buffer size with *b* for block (512 bytes) or *k* for kbytes of memory to use.

package

package uses the *bs=* amount for the *write* buffer and 10 percent, or 5120 bytes, whichever is more, for the *read* buffer. The maximum buffer size is 90 percent minus the program size of the available program memory on the system.

after=date Dumps only the files appearing after the specified date. This option is for incremental back ups of a system. Specify the date in the same string format as the UNOS **setdate** command, MMddHHmmYY.

MM Month in numeral format. (01 is Jan, 02 is Feb, etc)

dd Day of month.

HH Hour in 24 hour format. (9:00 PM is 2100)

mm Minutes.

YY Last two digits of the year. (1985 is 85)

-s Indicates silent mode. This option doesn't report every file and size to the terminal.

-help Specifies the standard UNOS usage function for the on-line description.

-q Specifies query. This option prompts the console with the file name and asks for a response of y or n. If you specify n, the file is not saved.

-c Adds up all the sizes of the files that would be dumped if this option was unspecified. Use this option to see how many volumes or how much space is required for an incremental backup. For example, the *after=* option.

-recover Allows the program to continue from disk read errors on files and directories. The I/O is done down to the block increment when an error occurs on a buffer fill and the block is retried. If the block is unreadable

- the area of the buffer that would have contained the block is NULLED (zeros written)
- the file name and logical block in the file is reported to the console
- the program continues with the file

If the error is on a directory

- the directory block is skipped
- the directory name is reported with a warning that some entries may not be saved

Use this option to save files from a disk with bad blocks.

-x Specifies the first name on the command line is the device to write the output to.

-nr Specifies no recursion. The directory tree does not dump any subfiles or directories of any named files.

-V Writes volume headers on each new volume. This option detects a disk mix-up.

NOTE: If this option is specified, the output is not cpio compatible.

ib=MMddHHmmYY (incremental backup) Provides for incremental backups of any files with a date/time that falls between the current time and the current time minus the option argument string. For example, *ib=00070000* backs up any files 7 days (1 week) old because 7 is indicated in the day field and the other fields are empty. When the 7 field is subtracted from the current date, the difference is a week.

-tape This option closes the device between volumes so that the device driver can write a file mark on the tape, and not latch the "tape changed" status. To use this option, you must specify the -x option also so the program knows the name of the device to open for the next volume.

EXAMPLES

To save some files to a tape:

```
#package bs=128k package.c package.doc package.dir  
>/dev/rct0 <cr>
```

To save file using a *take* file:

```
#list /release >takefile <cr>  
#package -x /dev/rcfd0 <takefile <cr>
```

package

To see how many blocks you need to do an incremental back up on a system:

NOTE: To ensure the files are saved make sure all the `/usr` and extra partitions are mounted.

```
#package after=0903090085 -c / <cr>
```

To backup all the `/usr` files to tape after a certain date:

```
#package bs=1024k after=0903090085 /usr >/dev/rct0 <cr>
```

To recover from the `/src` file system that developed bad blocks:

```
#package bs=512b -recover /src >/dev/rct0 <cr>
```

NOTE: This `-recover` option displays on the console disk controller errors as well as package messages of the file(s) with bad blocks. Error reporting may take some time on a really bad disk as this option tries to get every last valid readable block from the file system.

To backup a system that may take more than one tape:

```
#package bs=512k -tape -x /dev/rct0 /
```

DIAGNOSTICS

package prints a summary of the arguments on a command error. **package** also prints out errors when reading files with the `-recover` option. If an error occurs mid-volume on an output disk, restart with new media.

ALSO SEE

installation reads files from a tape or disk in an installation file format

debe copies a file or a piece of a file

NOTES

The **package** program along with the **installation** program is recommended as a replacement for the **backup** and **restore** programs.

package

Features that make **package** a good choice for backing up and restoring file systems include

- handling bad blocks
- using large buffers
- getting a listing of all files on a dump
- dumping while the machine is running
- restoring individual files by name or pattern
- UNIX cpio compatibilty between machines

package also helps OEM's make software releases that are compatible with the UNOS **installation** command.

If you are backing up or restoring, do so as admin. Otherwise, when you restore the file system, you will lose the files for which you did not have read permission.

USES

nothing

paste

paste combines files in columns

FORMAT

paste [*d*=*C*] [*file1*...*filen*]

DESCRIPTION

paste joins separate files by fields. It is the opposite of the **cut** command.

paste joins corresponding lines of files creating a single line of text with a single character delimiter separating each field. The result is displayed on standard output.

OPTIONS

d=*C* Where *C* (a single character) specifies the field delimiter used to separate the fields on output. The default delimiter is TAB.

EXAMPLES

This example uses 2 files:

col.1:		col.2:	
1		x	
2		y	
3		z	

```
$paste d=: col1 col2
```

```
1:x
```

```
2:y
```

```
3:z
```

DIAGNOSTICS

none

ALSO SEE

cut dissects files by fields or columns

paste

NOTES

Each file can have a maximum line length of 512 bytes. The maximum number of files that you can join together is 69.

USES

nothing

pmask

pmask displays the current file creation mode mask

FORMAT

pmask

DESCRIPTION

pmask prints the current file creation mode mask. This mask determines the actual modes of a created file.

OPTIONS

none

EXAMPLES

In this example, all modes are currently allowed.

```
$pmask
```

```
rwX rwX rwX
```

In the next example, others are denied read and write access, but group members are permitted read access:

```
#pmask
```

```
rwX r.X ..X
```

DIAGNOSTICS

none

ALSO SEE

setmask sets the file creation mode mask

NOTES

none

pmask

USES

nothing

#

performs special Command Interpreter functions

FORMAT

```
#a[g|l]
#b[g|l]
#d[g|l]
#e
#h
#l[g|l]
#p[g|l] [a|b]
#g
#s[g|l]
#x[name=value]
#v[on|off]
#?
```

DESCRIPTION

commands (sometimes called **splat** commands) are a special type of UNOS Command Interpreter command. They all begin with a pound-sign and perform a variety of function that let you alter the way the system appears. You use **splat** commands to rename other commands, create abbreviations for complicated command lines, and repeat your last command.

Because the **splat** commands are special, they do not have the functionality of regular UNOS commands:

- You cannot use pattern matching with **splat** commands.
- You cannot use **#e** to re-execute **splat** commands.
- You cannot execute any **splat** commands while in VED (although you can execute abbreviations).

Abbreviations let you assign your own label to an expression (a pathname or a filename) or command. Abbreviations are permanent; they exist until you change or delete them. They can be attached to a user (global) or they can be attached to a directory (local). Global abbreviations are in effect whenever you log in and wherever you are on the system. Local abbreviations only work in the directory that contains them; they will always override global abbreviations. When you create abbreviations, the

#

default is global. You can, however, define an abbreviation to be global ("g") or local ("l"). See the **NOTES** section for more details on local and globals.

The following list describes **splat** commands:

#a

defines an abbreviation that can be used "anywhere".

#b

defines an abbreviation that must appear at the "beginning" of a command.

#d

deletes an abbreviation.

#e

re-executes the last UNOS command. It will not re-execute a **splat** command.

#h

displays help about **splat** commands.

#l

displays all defined abbreviations. By default, #l displays global abbreviations; use #ll to display local abbreviations.

#p

temporarily defines or redefines abbreviations. (See the **NOTES** section for more details.)

#q

logs you out. It is the equivalent of ^D.

#s

changes the default (global or local) of **splat** commands. (See the **NOTES** section for more details.)

#x

displays and/or changes elements of the environment. (See the **NOTES** section for more details.)

#v

turns verbose mode on or off. Verbose mode displays every command line before execution. If a command line includes an abbreviation, it is expanded in the verbose display.

#?

displays help about **splat** commands.

OPTIONS

a expression abbreviation.

b command abbreviation.

g makes an abbreviation global.

l makes an abbreviation local.

ARGUMENTS

name=value used with **#v** to create or change a name-value pair in your environment.

on used with **#v** to turn verbose mode on.

off used with **#v** to turn verbose mode off.

EXAMPLES

This example shows using **#a** to create an abbreviation called *jy*, and then using the abbreviation with verbose mode turned on:

```
##a jy install/doc/work/july
#cd /usr/john/jy/manuals
{/usr/john/install/doc/work/july/manuals}
```

The following example uses **#b** to create an abbreviation that renames the **delete** command:

```
##b rm delete
```

The following example first creates one abbreviation called *mc*, then creates another abbreviation, called *L*, that uses *mc* in its definition.

#

l will display a list of files in multiple columns.

```
##b mc prfmt -nh l=1 -w80 -5
```

```
##b l do "list $r1 | mc"
```

Use **#x** to see your default environment:

```
##x
```

```
PATH=:/bin
```

```
HOME=/usr/dianne
```

```
TERM=z19
```

```
SHELL=/bin/command
```

This list describes the **splat** commands:

Use **#x** to change the *HOME*= environment:

```
##x HOME=/usr/sales
```

```
#cd
```

```
#pwd
```

```
/usr/sales
```

To see an abbreviation called **cp**, use the **#l** command and the abbreviation name as an argument (pattern matching does not work here; you cannot use asterisks):

```
##l cp
```

```
#b cp copy
```

To change the default to always list the local abbreviation, simply type

```
$#sl
```

To see the help file for the special commands, type either **#h** or **#?**.

##h

#a[g|l] name value - add new abbrev

#b[g|l] name value - add new begin abbrev

#d[g|l] name - delete abbrev

#e re-execute last non-# command

#h display complete help file

#? display complete help file

#l[g|l][name] - list abbrevs

#p[g|l][a|b] name value - push abbrev definition

#q exit command interpreter

#s[g|l] set default table for # commands

#x [name=value] display [export to] environment

#v [on|off] display [set] verbose mode

DIAGNOSTICS

none

ALSO SEE

command executes a command line or file

User's Guide to UNOS

NOTES

Abbreviations

Abbreviations can include variable substitution characters as part of their definition. This is useful when a command takes several arguments.

Use **\$1** to refer to the first argument, **\$2** to refer to the second argument, **\$3** to refer to the third argument, and so on.

To indicate multiple character substitutions, use **\$r#** where **#** indicates which argument to start with and the **r** indicates all the rest. For instance, **\$r1** says to use the first argument and all others. **\$r5** says start with the fifth argument and use all the rest, but ignore the first four.

#

If an abbreviation definition contains a substitution character, you must include the command line or expression in quotes, and precede it with the word "do". For instance:

```
#b print do "format dev=spin $r1 | lpr printer=fancy"
```

NOTE: Do not name an abbreviation *do*; *do* is a reserved word for variable substitution expansion.

Environment

Your environment is defined by the command search path, your home directory, your terminal type, etc. You can change these with the **#x** command. Type **#x** without an option to see your default environment. Use a *name=value* pair to change the environment.

Seven *name=value* pairs are standard elements of the environment:

LOGNAME=

PATH=

HOME=

TERM=

SHELL=

PROMPT=

MAIL=

LOGNAME= is the name with which you logged in.

PATH= refers to the search path, which is the route the "exec" subroutines take to find the commands you specify. The default path is the current working directory, the **/bin** directory, and then the **/usr/bin** directory. (Most of the UNOS commands are located in **/bin**.) Do not delete this pathname or you will not be able to execute UNOS commands. If you have your own commands, you can add their location as well.

To change your path, type **#x PATH=** and the pathnames, separated by colons. (Case is significant; type environment variables just as they appear.) An empty field means your current directory. For instance, to change the path to specify the **/bin**, the current directory, and the home directory, type:
##x PATH=:/bin::/usr/dianne

HOME= refers to where you return to when you type the change directory command, **cd**, without a directory name. Your default is the directory that matches your password entry. To change it, use the **#x** with the **HOME**= option to specify a new pathname.

TERM= specifies your terminal type. To get this information, the Command Interpreter reads the **/etc/ttys** file, which contains every terminal (tty) port on your system and the corresponding terminal type. For information about terminal descriptor files see the *System Administrator's Guide*.

SHELL= is whatever **/etc/passwd** uses for your shell, usually **/bin/command**.

PROMPT= is an environment variable which will be used by **command** to replace the **#** prompt, if given.

MAIL= is used by the UNIX **mail** command. See the UNIX Tools Documentation for more information.

#x considers all characters between the = and the newline to be the value, thus allowing spaces and tabs in the value.

Note that any environment changes you make with the **#x** command only last for your login period. When you log off, and login again, the default environment is in effect. To make a more permanent change to your environment, use the **#x** command in your *init* file. See the *User's Guide to UNOS* for details on the *init* file.

Push Abbreviations

You can temporarily define or redefine your abbreviations. This means that you can keep the same name but have it mean something else. This uses the concept of a stack: you can keep changing the meaning (pushing) and then revert to the previous meaning (pop) by deleting the abbreviation. You can change the meaning of an abbreviation as many times as you want but the change (stack) only stays in effect for the duration of your login session.

#

Abbreviations are pushed with the **#p** command. Use it with either the letter **a** (for expression abbreviation) or the letter **b** (for command abbreviation). To pop an abbreviation, use the **#d** command and the abbreviation name.

See the *User's Guide to UNOS* for more details and examples of pushing and popping abbreviations.

Global vs Local Abbreviations

When you create, delete, list, or push an abbreviation, the Command Interpreter assumes that you are referring to a global abbreviation. You can specify either global or local or you can change the default.

The command **#s** changes the default abbreviation: **#sl** makes all subsequent **splat** commands refer to local abbreviations and, conversely, **#sg** makes them refer to global abbreviations.

Note that changing the default table has nothing to do with the abbreviations themselves.

Global abbreviations are kept in a *.globals* file in your home directory. Everytime you login, your *.globals* file is read. You can edit the *.globals* file as you would any other file, but any abbreviations you add or change will not be acknowledged until you login again. For an immediate change to the *.globals* (or *locals*) file, use the appropriate **splat** command.

Local abbreviations are kept in a *locals* file in the directory in which they are created. You can have as many *locals* files as there are directories. Each time you change to a directory containing a *locals* file, that file is read and updated, regardless of who created it.

If you have a local abbreviation with the same name as a global abbreviation, the local abbreviation will be executed when you are in the correct directory. Local abbreviations *always* override global abbreviations; if you change to another user's directory, their local abbreviations will override your global abbreviations.

#

USES
nothing

prfmt paginates files for printing

FORMAT

prfmt [options] file1..n

DESCRIPTION

prfmt produces a formatted version of one or more files on standard output. The output is separated into pages headed by the file's modification date, the name of the file, and the page number.

OPTIONS

The options apply only to files following the option on the command line, and can be reset between files.

-b Brief

Prints one line each of the header and footer. Counting blank lines, only six lines are subtracted from the total page size.

c=author Places Copyright (c) year author instead of **-f1**.

-l' Places a formfeed after the file.

f1=footer First line of footer.

f2=footer Second line of footer.

f3=footer Third line of footer.

h=argument Uses the specified argument instead of the file name.

(If you use **prfmt** as a filter, it will not have a name unless you specify the **h** option.)

l=# Takes page length to be the number of lines instead of the default 66.

prfmt

- `-m` Prints all files simultaneously, each in one column.
- `-#` Produces numbered column output of each file sequentially.
Any number between 1 and 69 can be specified.
- `+#` Begins printing with page number.
- `-nb` No brief
Prints three lines of the header and footer. Counting blank lines, a total of ten lines are printed. This is the default.
- `-nh` No header or footer.
- `-pb` Page break.
Does not print a header or footer, but leaves a blank line at the top and bottom of the page. This lets you paginate a file without a header or footer.
- `s=C` Separates columns by zero, one or more characters instead of white space. To specify no characters, type `s=`. The default character is TAB. Do not use control characters as delimiters.
- `w=#` Takes the page width to be the number instead of the default 132.

EXAMPLES

To produce a five-column listing of a directory's files, type
`@>list | prfmt w=80 -5 l=1 -nh`

DIAGNOSTICS

prfmt prints error messages for impossible cases, such as a line of 10 characters cannot have 11 columns.

ALSO SEE

cat concatenates files
copy duplicates files
lpr queues printing requests
p paginates files and displays non-printing characters

NOTES

When using **prfmt** to display text on the screen, specify the width as 80, and the length as 24.

If you use arguments that consist of several words, enclose them in double quotes.

USES

nothing

print_bad_blocks

print_bad_blocks displays bad block list

FORMAT

/sys/print_bad_blocks device_path1...device_pathn

DESCRIPTION

print_bad_blocks extracts, formats, and prints on standard output, the list of locations stored in the bad block record on the specified disk devices.

OPTIONS

none

EXAMPLES

```
#/sys/print_bad_blocks /dev/rccall
```

```
" Bad block list for 16M Winchester disk /dev/rcc0.  
" 3 defects * 16 blocks/defect = 48 unusable blocks.  
" 32720 remaining usable blocks.  
" Cylinder #      Head #  
           16         0  
           17         0  
           53         1
```

DIAGNOSTICS

Standard file system errors are reported.

ALSO SEE

write_bad_blocks creates/overwrites bad block list on disk

NOTES

The output from **print_bad_blocks** is in a suitable format to be used as input to **write_bad_blocks**. It is useful to save the list in the event the disk record containing the bad block list is

print_bad_blocks

overwritten.

USES

nothing

print_partition

print_partition displays the partition record for a hard disk

FORMAT

/sys/print_partition device

DESCRIPTION

print_partition displays the partition map stored on the specified device, if any. The output is in a format suitable for use as the input file for **write_partition**. The partition map is stored in the first physical block of a disk, so the device argument should name a partition that begins at the beginning of the disk. By convention, this is the first partition in the map (**/dev/rcq0**, **/dev/rcd0**, etc.).

OPTIONS

none

EXAMPLES

The following would be the output from a normal **cq0** partitioning:

```
#/sys/print_partition /dev/rcq0
"partition      offset      limit
0                0          2000
1             2000          4000
2             4000         24000
3            24000         44000
4            44000         64000
5             4000         44000
6            64000         68000
@>
```

DIAGNOSTICS

none

print_partition

ALSO SEE

write_partition creates/overwrites the partition record for a hard disk

NOTES

none

USES

nothing

print_pvh

print_pvh displays a physical volume header from a disk

FORMAT

/sys/print_pvh device

DESCRIPTION

print_pvh reads a physical volume header from the specified disk device. This allows the user to determine what sort of device this is supposed to be, according to the volume header, and the resulting output can be modified and fed back to the **write_pvh** command.

print_pvh takes the name of a disk device to read a physical volume header from as an argument. It writes the physical volume header on standard out, after converting it to a human readable form.

OPTIONS

-v prints all the defined fields in the PVH

EXAMPLES

To read the physical volume header from **/dev/rcd0**, the following command would do
#/sys/print_pvh /dev/rcdall

```
"32M_Winchester"  
"/dev/rcd0"  
cylinders:512  
heads:8  
sectors:32  
bytes:256  
reduce:128  
stepperperiod:0  
stepwidth:3  
stepmode:0  
controller:  
interleave:2
```


print_pvh

```
recovery:0
option 1:0
option 2:0
option 3:0
```

DIAGNOSTICS

Could not get status of '%s'.
The specified file cannot be accessed.

Device is not a disk
The specified file cannot have a pvh.

Cannot open device '%s'.
The specified device cannot be opened.

No pvh on '%s'.
The disk has no pvh.

No device specified.
Incorrect command syntax.

Invalid option '%s'.
Incorrect command syntax.

Cannot read disk.
The device cannot be read.

ALSO SEE

write_pvh places a physical volume header on the specified disk device.

UNOS System Concepts Manual

NOTES

This command is not applicable to floppies or tapes.

print_pvh

Always use the raw disk device. Pvh files include

- ca
- cb
- cc
- cd
- cj
- ck
- cj.dex
- ck.sec
- cm
- cq
- xe

USES

nothing

printec

printec prints eventcount values

FORMAT

printec [options] event_count_name1....event_count_namen

DESCRIPTION

printec displays on the terminal the value of the eventcount(s).
The print format is

Eventcount "foo": counter one is 10, counter two does
not exist counter two is 15.

OPTIONS

Note that these options are provided to make **printec** compatible with other UNOS commands. *-help* Prints to standard error a few lines which list all the options which the **printec** command will accept.

- silent* Tells the **printec** command not to print the exit message(s).
- query* For each eventcount, **printec** is asked to print, the *-query* flag will cause **printec** to ask the user if he really wants that eventcount printed.
- interactive* Causes the **printec** command, after acting upon any eventcounts given on the command line, to prompt the user for more eventcounts.
- verbose* Accepted and ignored.

EXAMPLES

```
#makeec Count  
#printec Count  
Count .1:0 .2:0]
```

printec

DIAGNOSTICS

can not open ec <eventcount_name>

ALSO SEE

advance increments eventcounts

makeec creates eventcount file system entries

NOTES

none

USES

nothing

printenv

printenv prints out the values of environment variables

FORMAT

printenv [*list_of_names*]

DESCRIPTION

printenv prints out the current process environment variable table in the form
variable=value

OPTIONS

list_of_names If the name exists in the environment, it will be displayed. If no names are given, **printenv** will display the entire environment.

EXAMPLES

```
$printenv
PATH=:/usr/john/bin:~/bin:/usr/bin:
HOME=/usr/john
TERM=z29a
SHELL=/bin/command
PROMPT=->
PS1=-)
ROGUEOPTS=name=Gorble the Slayer,fruit=cumquat,noterse

$
```

DIAGNOSTICS

none

ALSO SEE

nothing

printenv

NOTES

Each different command interpreter (**command**, **sh**, **csh**) has different conventions for printing and creating environment variables.

USES

nothing

printlog

printlog displays system error messages

FORMAT

/sys/printlog

DESCRIPTION

printlog extracts system error messages from a buffer in the kernel, formats them, and writes them to standard out. The first messages printed may be quite old, as they hang around until they are read (i.e., by **printlog**) or until they are overwritten when the kernel has to wrap around in the buffer. After **printlog** catches up with any old messages, it waits for new messages to appear.

Currently, only device drivers put error messages into the system log buffer.

OPTIONS

none

EXAMPLES

To keep track of all errors:
#printlog > logfile &

The output will look similar to

```
type 0x4200 source 0x4580 CC1: Error 0xa3 retried on sector 56049
type 0x4200 source 0x4580 CC1: Error 0xa3 retried on sector 56049
type 0x4200 source 0x4580 CC1: Error 0xa3 retried on sector 56049
type 0x6200 source 0x4580 CC1: UNIT 0 SECTOR 56049 ERROR 0xa3
type 0x4200 source 0x4580 CC1: Error 0xa3 retried on sector 56049
type 0x4200 source 0x4580 CC1: Error 0xa3 retried on sector 56049
type 0x4200 source 0x4580 CC1: Error 0xa3 retried on sector 56049
type 0x6200 source 0x4580 CC1: UNIT 0 SECTOR 56049 ERROR 0xa3
type 0x4200 source 0x4580 CC1: Error 0xa3 retried on sector 0
type 0x4200 source 0x4580 CC1: Error 0xa3 retried on sector 0
type 0x4200 source 0x4580 CC1: Error 0xa3 retried on sector 0
type 0x6200 source 0x4580 CC1: UNIT 0 SECTOR 0 ERROR 0xa3
```


printlog

DIAGNOSTICS

none

ALSO SEE

The following subroutines:

log_errori()

log_errorb()

readerrlog()

NOTES

printlog does nothing if the error logging feature is not enabled, which is done by setting a configuration parameter ("size of system error log buffer") to some reasonable value such as 100. This value is zero in the standard config file, which prevents logging.

See the **/include/sys/errlog_format.h** for an explanation of the **printlog** formats.

USES

/include/sys/errlog_formats.h

This file contains C definitions of error log codes with corresponding error messages.

profile

profile produces a function histogram

FORMAT

profile [-sys][-v][bar=#][o=file][ld=file] *command*

DESCRIPTION

profile uses the **logger** and **hist** programs to produce a function histogram for the *command*.

OPTIONS

-sys Includes supervisor mode entries for the *command* as well as user mode entries.

-v Verbose mode.

Prints all entries for the *command* as they are read. The format is:

entry number; addressing mode(S or U); pc;

followed by the function (if it can be determined).

bar=# Sets the scaling factor for the histogram bars to the specified number. The default value is 10. Bar=0 will suppress printing of the bars, showing only the function name and the reference count.

o=file Specifies the output file rather than standard out.

ld=file Specifies the file for the logger output. The file is not deleted. The default is to create a temporary file and then delete it when the **hist** program is finished.

EXAMPLES

This command line produces user and supervisor mode histograms for the *command testprog*.

@>profile -sys testprog

profile

DIAGNOSTICS

profile generates an error message if it cannot open a needed file or if it cannot start the logger, the hist, or the specified command.

ALSO SEE

hist constructs a histogram of function usage

logger collects timing information

NOTES

profile does no pathname searching. If the command is not in the current directory specify the complete pathname. Since it calls the subroutine **exec_env** directly when starting the command, it bypasses the system's search rules. This ensures that if the command executes, the **hist** program can find the symbol table.

The **-ld** option is the only option processed by **profile**; all the other options are passed to the **hist** program.

USES

/tmp/<pid>.profile

prtansi understands ANSI printer control characters

FORMAT

prtansi [vfu=name] file1..n

DESCRIPTION

prtansi expects extended American National Standards Institute (ANSI) printer control characters as the first character of every record in every file being formatted. The files listed on the command line (or standard input) are formatted on standard output.

/etc/vfu

prtansi uses the *vertical format units* (vfu) contained in the **/etc/vfu** file. Each record in **/etc/vfu** contains one format unit. Each format unit is identified by a unique format name. A sample record is included here for reference.

Records in **/etc/vfu** contain multiple fields. The fields are delimited by single colons (:). For example, the record
hello: 5:: bye
contains four fields: 'hello', '5', '', 'bye'.

The first field, 'hello' in the example above, is considered to be the name of the format unit.

The second field is the physical page size (which must be an integral number of printlines long).

The third field is the logical page size, and reflects the number of lines you want printed on each page. For example, standard typewriter paper has 66 physical print lines, but most people type on only 60 of them. The top and bottom three lines are left blank to improve readability.

Both the physical and logical page size fields consist of a number followed by an optional comment.

prtansi

Because a request to move to logical line number one is very common, many printer manufacturers incorporate high-speed skip to top-of-form hardware in their printers. A form-feed control character invokes the high-speed skip.

In a UNOS vertical format unit, anywhere a line number or line count can be specified, the two characters 'ff' can be substituted. The 'ff' token means 'issue a form-feed'. The result of the form-feed is hardware dependent. However, wherever the printer moves is considered logical line one.

If a physical pagesize field is left blank, 'ff' is assumed.

If a logical pagesize field is left blank, it is assumed that the logical pagesize equals the physical pagesize.

You cannot comment the physical or logical pagesize fields without first specifying a pagesize or 'ff'.

/etc/vfu Channels

Format record fields four through fifteen are considered optional. Each defines a separate print channel. Twelve print channels are available numbered one through twelve. The order in which the print channels are defined is unimportant.

Each print channel field consists of a channel number followed by an unlimited number of logical line numbers. The numbers can be delimited by spaces, commas, or tabs. Commas before the channel number are not allowed.

The following is an example of a print channel field

```
: 3 4,5 2,6,ff,1 :
```

It declares channel three as containing logical line numbers 4, 5, 2, 6, ff, and 1, in that order. If a skip to channel 3 command is received, the UNOS soft vertical format unit will move the paper to logical line number 4. A second channel 3 command will move the paper to line 5. Similarly, line numbers 2, 6, ff, and 1 will be cycled through. If more channel 3 commands are received, the pattern will repeat.

A backslash character specifies that you would like to follow the channel definitions with a comment.

OPTIONS

The argument *vf**u*=*name* specifies the vertical format unit to be used. The *vf**u*'s are contained in file */etc/vfu*. If a *vf**u* is unspecified, *vf**u*=*default* is used.

EXAMPLES

To print ANSI printer file *print.out* using *vf**u* *paychecks*, type
`$prtansi vf=paychecks print.out`

To print ANSI printer file *from.ibm* using the default *vf**u*, type
`$prtansi from.ibm`

The following are examples of UNOS vertical format units:
*default:::1 ff * suggested default for all systems
payroll:15::1 10 13: 2 9,12
statements:95:90:5 50,7,ff,50:1 2,3,ff,89:

DIAGNOSTICS

none

ALSO SEE

nothing

prtansi

NOTES

<u>Code</u>	<u>Action Before Printing a Line</u>
b	spaces one line ('b' is the blank character)
()	spaces two lines
-	spaces three lines
+	suppresses space (overprint this line)
1	skips to channel 1
2	skips to channel 2
prtansi	skips to channel prtansi
4	skips to channel 4
5	skips to channel 5
6	skips to channel 6
7	skips to channel 7
8	skips to channel 8
9	skips to channel 9
A	skips to channel 10
B	skips to channel 11
C	skips to channel 12

All other characters are interpreted as *b* and no error indication is returned.

USES

/etc/vfu to find vfu's.

ps displays process status

FORMAT

```
ps
[-all][-fphw][-loc][-long][pid=#][-pp][-short][-sys][username
username...]
```

DESCRIPTION

ps prints information about active processes.

Information about active processes is displayed in fields (as follows). Note that if a specific field is not always displayed, the flag that causes it to be displayed follows in parentheses.

PID The process identification number.

USER The name of the user who initiated the process.

FLG Process state flag. These flags are ANDED into one word.

CODE	Meaning
0x1	The process in swapped out.
0x2	The process is doing kernel I/O.
0x4	This is a kernel process.
0x8	The process is doing raw I/O.
0x10	Internally used flag.
0x20	The processes is doing an interruptable wait (I/O, eventcounts).
0x40	The process is being debugged.
0x80	The process is being traced.
0x100	The processed is locked into memory.
0x200	Internally used flag.
0x400	The processes is a standard 3 segment process, not a 2 segment processes.
0x800	The processes is waiting on another process.

STAT The state of the process:

ps

CODE	Meaning
EXIT	The process is exiting.
FORK	The process is forking
RDY	The process is ready to run.
RUN	The process is running (always ps).
SSPD	The process is being suspended.
STOP	The process is suspended.
WAIT	The process is waiting for some event (see WAITFOR).

PRI Current priority.

WAITFOR What the process is waiting for.
It can recognize some of the types of eventcounts a process can wait on. Eventcounts that it cannot recognize are printed as hex numbers.

CODE	Meaning
event	Normal event count.
pipe	Pipe event count.
lock	File lock eventcount.
timeout	Clock eventcount.
tty	tty device.

Kb The size in Kbytes of the core image of the process.

SYS The amount of system time used in seconds and tenths.

USER The amount of user time used in seconds and tenths.

PROGRAM Program name with its arguments,
"-" means Command Interpreter (/bin/command),
"+" means running **admin** on top of
the user login.

NOTE: The name and argument list is limited
to 33 characters unless the *-long* option is used.

text Beginning and end physical address of the text segment. (*-loc*)

data Beginning and end physical address of the data segment. (*-loc*)

stack	Beginning and end physical address of the stack segment.(-loc)
MAX	Maximum priority. (-long)
MIN	Minimum priority. (-long)
CHLD	Child system time (in seconds and tenths). (-long)
TIMES	Child user time (in second and tenths). (-long)
AGE	Time (in seconds and tenths) that processes has been on its present queue. (-long)
LU	Number of open Logical Units. (-long)
INT	Outstanding process interrupts. (-long)

CODE

Meaning

0x80	This process will be preempted.
0x40	This processes will exit.
0x20	This process will suspend.
0x10	Indicates that process will be preempted.
0x08	Flag used by kernel tracing routines.

SIGNALS Outstanding signals.
UNOS signal numbers correspond to
bits in this word. (-long)

If no names are specified, **ps** prints the process status of the user. If names are specified, **ps** prints the status of those users.

OPTIONS

If no options are specified, **ps** displays the status of the users' processes.

-all Displays all processes, not only your own.

-fphw Displays hardware floating point status and registers.

ps

-loc Displays the sizes and locations of the text, data and stack segments.

-long Displays much more information about each process.

pid= Displays information only about this processes.

-pp Displays parent process ID

-short Displays parentage

-sys Displays the system processes (swap and any other kernel process)

username Displays information only about processes that belong to this user.

EXAMPLES

The first example shows using **ps** with the *-all* option.

```
#ps -a
```

PID	USER	FLG	STAT	PRI	WAITFOR	Kb	SYS	USER	PROGRAM
1	admin	21	WAIT	91	child	27	1.2	0.0	start
21	rfm	20	WAIT	95	child	46	2.5	1.1	-
10	admin	20	WAIT	80	timeout	4	9.7	0.0	/sys/update
22	dan	20	WAIT	95	child	46	4.8	2.6	-
23	allen	20	WAIT	94	child	62	8.3	21.4	-
24	admin	21	WAIT	95	tty	28	0.4	0.0	login /dev/tty5
127	dan	20	WAIT	99	tty	48	2.8	0.8	ved shell
123	allen	2	RDY	73		5010.1	5.2		ved change.c
131	rfm	0	RUN	79		29	1.6	0.5	ps -a

This example shows using the *-long* and *pid=* options. (This display appears on one line on your screen.)

```
#ps -long pid=37
```

PID	USER	FLG	STAT	PRI	MAX	MIN	Kb	SYS	USER	CHILD	TIMES	AGE
37	bob	C20	WAIT	95	100	0	72	2.3	1.1	15.5	22.4	1.3

LU INT SIGNALS

3	0	0
---	---	---

WAITFOR 23890 child

PROGRAM -

Here we look at bob's process #37. The FLG field shows that the process is doing an interruptable wait on another process and is a 3 segment process. The other fields describe additional information about process #37.

The next example shows using the *-fphw* option.

```
#ps -fphw<cr>
  PID USER   FLG STAT PRI WAITFOR Kb   SYS  USER PROGRAM
   37 bob    C20 WAIT  94 child  72   2.2   1.0 -
0   0   0   0   0   0   0   0   0
comreg =    0, stcreg = 5FAC
HALTR RESET IDLE
```

Line 0 Titles.

Line 1 the standard *ps* line.

Line 2 the 8 registers of the SKY FFP board.

Line 3 the contents of the Command and the Status and Control registers.

Line 4 the status flags of the SKY FFP.

Refer to the FFP manual for an explanation of these flags and register contents.

NOTE: There was no process using the *ffp* when this *ps* was executed.

The last example shows using the *-loc* option.

```
#ps -loc<cr>
  PID USER   FLG STAT PRI WAITFOR Kb   SYS  USER PROGRAM
   37 bob    C20 WAIT  99 child  72   2.2   1.1 -
text: 0xA4000-0xADFFF, data: 0xEF000-0xF4FFF,
stack: 0xF5000:0xF6FFF
```

Here we see process #37 again. This time the starting and ending addresses of the process are displayed in hexadecimal notation. These are physical (bus) addresses.

ps

ALSO SEE

kill sends a signal to processes

resume resumes execution of a suspended process

suspend halts execution of a process

NOTES

ps takes real time. The various process states are taken over time, not instantaneously.

ps can take quite a long time. When processes are swapped out, **ps** reads the swap device to get the information (from the *u_area*).

USES

/dev/kmem

/etc/passwd

pwd

pwd displays name of working directory

FORMAT

pwd

DESCRIPTION

pwd prints the working (or current) directory, from where the request was initiated, as an absolute path from the root.

OPTIONS

none

EXAMPLES

In this example, **doc** is the name of the current directory.

```
#pwd
```

```
usr/john/doc
```

DIAGNOSTICS

none

ALSO SEE

cd changes working directory

NOTES

none

USES

All the directories above the current directory are searched as a path to the root.

qfix repairs corrupted queues

FORMAT

`/sys/qfix [-help] queue`

DESCRIPTION

qfix repairs corrupted queues and allows normal processing to continue. System crashes and other calamitous circumstances can occasionally leave queues in an inconsistent state. Such queues will not properly distribute packets for processing, and they often remain stuck, blocking on some request made by the calling program.

qfix the name of a queue as its argument. It examines that queue and puts it into a consistent state.

OPTIONS

-help Prints, to standard error, a few lines listing all the command options **qfix** accepts.

EXAMPLES

If the queue of requests for the elk-tracking radar dish has somehow become corrupted, and no requests are being processed, use the command:

```
#qfix /etc/_queues/dish.elk
```

DIAGNOSTICS

can not open queue

The given queue is inaccessible or nonexistent.

ALSO SEE

lpmaint maintains printer queue

qmaint controls processing of a queue

qprint lists the contents of a queue

qfix

NOTES

You must have proper access to a queue to fix it.

Queues usually become inconsistent when a process dies unexpectedly while a queue is open. It is harmless to fix a queue that is not corrupted, so when in doubt, **qfix** it anyway.

Never fix a queue that is in use.

USES

nothing

qmaint controls processing of a queue

FORMAT

qmaint	[-help]	queue	-kill packet_number
qmaint	[-help]	queue	-stop packet_number
qmaint	[-help]	queue	-resume packet_number

DESCRIPTION

qmaint allows users to control or prevent the processing of packets on a given queue. With this command, the user can take a packet off the queue, suspend the processing of a packet, or resume the processing of a suspended packet.

OPTIONS

- kill** Causes the named packet to be removed from the given queue. The packet will not be processed and it will disappear from the queue. If the packet is being processed, it will be stopped.
- stop** Causes the processing of the selected packet to be suspended until further notice. If the packet is not being processed, it will not be until it is resumed.
- resume** Allows a stopped packet to run again.

EXAMPLES

To prevent packet 717 on the washing machine queue from ever being processed, use the command:

```
#qmaint /etc/wash.machine -kill 717
```

DIAGNOSTICS

bad argument

command syntax error

packet not found

The given packet isn't on the specified queue

qmaint

bad packet number

The number given is not a valid packet number

can not open queue

The given queue is inaccessible or nonexistent

multiple flags specified

Only one operation may be done at a time

ALSO SEE

lpmaint maintains printer queue

qfix repairs corrupted queues

qprint lists the contents of a queue

NOTES

none

USES

nothing

qprint

qprint lists the contents of a queue

FORMAT

qprint [-help] queue

DESCRIPTION

qprint shows the contents and status of a given queue. It lists each packet on the queue, and prints pertinent information about it to the standard output.

When invoked, **qprint** prints, for each packet, the packet number, its priority, the state of the packet, and the contents of the packet.

OPTIONS

-help Prints, to standard error, a few lines listing all the command options **qprint** accepts.

EXAMPLES

If there is a queue of telephone requests, a **qprint** of it resemble:
`$qprint /etc/_queues/tel.wakeup`

```
/etc/_queues/tel.wakeup:
req pri state      packet
24276 0 working    admin    call 9-1-555-1234
24277 0 waiting    admin    call 9-1-555-4321
24278 0 waiting    rfm      hangup -time 5
24279 0 waiting    rfm      call 9-1-555-1122
24280 0 waiting    rfm      listen
```

DIAGNOSTICS

can not open queue

The given queue is inaccessible or nonexistent

qprint

ALSO SEE

lpmaint maintains printer queue

qfix repairs corrupted queues

qmaint controls processing of a queue

NOTES

Corrupted queues sometimes look very peculiar.

USES

nothing

quicksort

quicksort sorts the contents of files

FORMAT

quicksort [options] file1...filen

DESCRIPTION

quicksort arranges all the lines from the specified files according to the options you have selected and sends the results to standard output.

OPTIONS

-b Ignores leading blanks when comparing records.

bs=# Specifies the size of the internal buffer. The default buffer size is 5000 bytes. A larger value makes for a faster sort. If the value you specify is too small, **quicksort** will increase it.

-c All character comparisons are case insensitive, so that, for example, both $A < Z$ and $a < z$.

d=C Specifies the field delimiter. The character **C** (a single character) denotes the field delimiter. If you do not specify a delimiter, the default is a TAB. To specify a blank as a delimiter, enclose the entire option in quotes; i.e., **d=** .

f=#n,#n Sorts on specified fields separated by a delimiter defined with the **d=** option. Fields are by default arranged in the ASCII collating sequence. The optional character **n** specifies either *a* (ascending) or *d* (descending) in the result. Each field can be sorted independently, i.e., field1 ascending and field2 descending. The letter *n* can follow the field and specifies a numeric sort for that field.

-n Ignores leading white space and sorts on the first numeric characters. It recognizes decimal points as part of the numeric characters.

quicksort

- ns Not-stable sort. Does not necessarily retain the original order of identical items.
- r Reverses the sense of the sort. The file is sorted in the reverse ASCII collating sequence.
- u Gives a stable, unique sort. If more than one occurrence of an item appears in the file being sorted, use this option to see the first, and only the first, occurrence of the item.

EXAMPLES

The first example uses the following file, called *test*:

```
admin:zwe7qz.2:0:3:Q:/:/bin/command
jeff::05:1:X:/usr/jeff:/bin/command
jim::10:1:Q:/usr/jim:/bin/command
rfm:2a1OWmWd:88:1:Q:/usr/rfm:/bin/command
```

This command line sorts on field three, arranging the lines in descending order with the output delimiter being a colon, `:`.

```
$quicksort f=3d d=: test
rfm:2a1OWmWd:88:1:Q:/usr/rfm:/bin/command
jim::10:1:Q:/usr/jim:/bin/command
jeff::05:1:X:/usr/jeff:/bin/command
admin:zwe7qz.2:0:3:Q:/:/bin/command]
```

The next examples use the following file, called *foo*:

```
5.6
10.95
99.23
84.513
9.5
```

First, the file is sorted without any options. Note that in this example, the leading white space determines the order of the sort.

```
$quicksort file
84.513
9.5
10.95
5.6
99.23
```

quicksort

In the next example, the leading white space is ignored and the numbers are arranged in ascending order, including the decimal points.

```
$quicksort -n file
```

```
5.6
  9.5
10.95
 84.513
99.23
```

DIAGNOSTICS

none

ALSO SEE

nothing

NOTES

The *-r* and *-n* options are global and are only used in lieu of the *-f* option. By default, **quicksort** arranges the records in ascending ASCII collating sequence with the entire line used in the comparison.

Unless you specify the *f=*, *-b*, *-n*, or *-c* options, the *-ns* option is assumed.

USES

/tmp/st#.pid

reboot

reboot reboots the operating system

FORMAT

`reboot [os= ][iop= ][config= ][-noconfig][-verbose]`

DESCRIPTION

reboot reboots the operating system. It attempts an orderly shutdown, that is, it marks all mounted file systems as unmounted and updates the file system. If you do not specify any options, it uses the default operating system, iop, and configuration file. You must be the system administrator to use **reboot**.

OPTIONS

NOTE: All flags can be specified with their initial letter.

os= The operating system. The default is */sys/unos*.

iop= The IOP file. The default is */etc/iop*.

config= The configuration file. The default is */etc/conf*.

-noconfig The system does not load a configuration file, but instead prompts you for the root device.

-verbose Verbose. Gives you the name and size (in bytes) of each file that it uses. For example when loading */sys/unos* you will see

os '/sys/unos' # bytes

EXAMPLES

To reboot the standard file system:

```
$reboot
```

To specify the configuration file *cb2conf*:

```
$reboot config=cb2conf
```

reboot

DIAGNOSTICS

reboot prints various self-explanatory error messages.

ALSO SEE

nothing

NOTES

You must have enough memory to hold **reboot**, the operating system code, the IOP code, and the configuration file. Although the actual minimum amount of memory depends on how you have configured your system, a safe estimate is that systems of less than a quarter of a megabyte will not be able to use **reboot**.

Note that this command can also be used to boot in standalone software.

USES

nothing

reltocoff

reltocoff converts a file from the old UNOS object format file to COFF format

FORMAT

reltocoff old_format_file new_coff_file

DESCRIPTION

reltocoff converts files from the old UNOS object format to the new UNOS object format (COFF).

OPTIONS

none

EXAMPLES

Convert *f.j*, a file in the old UNOS object format, to the new COFF format. The new object code is placed in the file *f.o*.

```
$ reltocoff f.j f.o
```

DIAGNOSTICS

Self-explanatory

ALSO SEE

cofftorel converts a COFF format object file to the old UNOS object file format

NOTES

none

USES

nothing

repeat

repeat executes a command repeatedly

FORMAT

repeat [count=#] [delay=#] command

DESCRIPTION

repeat causes the command line to be executed continuously until either the specified number of iterations has completed or it is stopped (by typing ^C).

repeat can use any command and its arguments. However, special characters that are interpreted by the Command Interpreter must be quoted in order for them to be correctly passed to the **repeat** command.

Simple command lines need not be quoted because **repeat** combines all the trailing arguments to form the command that it executes.

OPTIONS

All options can be abbreviated by their first letter.

count=# Executes # times.
The default is forever.

delay=# Waits # seconds between repetitions.
The default is zero.

EXAMPLES

This command line makes the file *x* 64 times longer by replicating its contents. `$repeat count=6 "cat x x > y; move y x"`

The next command line spawns a background process to display the time of day, once a minute.
`$repeat delay=60 date&`

repeat

The next command line spawns four background processes executing the same command.

```
$repeat count=4 "command acceptance.test&"
```

DIAGNOSTICS

none

ALSO SEE

nothing

NOTES

repeat uses the program **command** to execute the command line. If there is a file of this name in the working directory, peculiar consequences result.

USES

command to execute the command.

restore

restore recreates a file system from a backup

FORMAT

restore [-r|-file|-fd][-l] *input_device* *arg_list*

DESCRIPTION

restore restores file system dumps created by the **backup** program. It can restore the entire file system or selected files from the dump. The backup volumes are read from *input_device*. The format of *arg_list* varies according to the option selected.

OPTIONS

- r Does a full restoration of a backup file system. **arg_list** should be the device onto which the file system will be restored. The receiving device must be a file system, that is, contain a valid system header and be at least as large as the backed-up file system. Take care when using this option, since it will overwrite any files already on the device. See the NOTES for the procedure for restoring incremental backups.
- file Restores by pathname one or more files from the backup. **arg_list** should be a list of pathnames to restore. Files are restored in the current working directory with a name of <file_number>.restore. A message is printed on standard out as each file is restored giving the original name and the restored name. Pathnames must be absolute and pattern matching characters are not recognized. If the file system being restored was a mounted file system, the path to the mounted directory should be omitted.
- fd Restores files from the backup by file number. **arg_list** should be a list of file numbers to restore. Files are restored in the current working directory with a name of <file_number>.restore. A message is printed on standard out as each file is restored giving the restored name. This option is not as useful as the -file option, but if the file number of the file to be restored is known it is much faster.

restore

-l Lists on standard output the information from the backup header. No **arg_list** is required for this option and it can be combined with other options. This is the default option in lieu of any other options. It displays the time of the backup, the backup id, the time of the oldest and newest files in the backup, and the number of volumes in the backup.

EXAMPLES

For full restoration of a backup from cfd0 onto cc0, type

```
$restore -r /dev/rcfd0 /dev/rcc0
```

To restore the file **/etc/ttys** from a backup, type

```
$restore -file /dev/rcfd0 /etc/ttys
```

To restore file numbers 17 and 47 from a backup, type

```
$restore -fd /dev/rcfd0 17 47
```

The next command line displays header information from a backup.

```
$restore -l /dev/rcfd0
```

DIAGNOSTICS

There are several possible error messages concerned with errors on the devices or failure to find a file number or pathname in the backup. They are all self-explanatory.

ALSO SEE

backup makes a backup copy of a file system

NOTES

The procedure for restoring a file system from a full backup plus a set of incremental backups is as follows:

1. Make sure your restore device is unmounted and no other programs can access it during the procedure.
2. Use **makefs** to create a new file system of the appropriate size on the device.

restore

3. Use restore with the **-r** option to restore the full backup.
4. Use restore with the **-r** option to restore each of the incremental backups in order, from the oldest to the newest.
5. Use **fscheck** to verify that the backup was restored correctly.

Under some conditions a file that was deleted during the period of incremental backups might be restored with no directory entry. **fscheck** will flag this condition with an **allocated but not referenced** message. **fscheck -fix** will correct it.

When restoring individual files by pathname, **restore** must trace each directory in the pathname through the backup. In a multi-volume backup this means that it will first request volume 1 but then may request volumes in any order.

USES

nothing



resume

resume resumes execution of a suspended process

FORMAT

resume pid#

DESCRIPTION

resume continues execution of a process which has been suspended, either intentionally, with the **suspend** command, or through an error in the program.

OPTIONS

none

DIAGNOSTICS

can't resume #

ALSO SEE

command executes a command line or file

kill sends a signal to processes

ps displays process status

suspend halts execution of a process

NOTES

Resuming a process suspended because of a trap (e.g. an illegal instruction) results in the process terminating with the appropriate return code.

Resuming a process from the command level causes the Command Interpreter to wait for it to complete. However, if **resume &** is used, the Command Interpreter does not wait; it prints the process id of the resume command itself (this number should be ignored).

Resuming a process does not resume the process's children.

resume

USES

nothing

setdate

setdate sets system date and time

FORMAT

setdate MMDDHHMM[yy]

setdate hhmm

DESCRIPTION

setdate sets the system clock. The system administrator should set the date when the system is started up. The date is typed **monthdayhourminute**, with the year optional (default is the year last set). To set the time for the current date, only the hours and minutes need to be specified (the 24 hour day is used).

OPTIONS

none

EXAMPLES

To set the date to Nov 15 (of the current year) 9:30 a.m., type
\$setdate 11150930

DIAGNOSTICS

setdate: Access permission denied. Could not set date.

Only the system administrator can set the date.

ALSO SEE

date displays the system date and time

NOTES

The system date and time is maintained as the number of seconds since 00:00 Jan 01, 1980. Therefore, times before that cannot be displayed or set.

setdate

USES

nothing

setmask

setmask sets the file creation mode mask

FORMAT

setmask [+ -]{rwx [+ -]{rwx} [+ -]{rwx}

DESCRIPTION

setmask sets the file creation mode mask. This mask is used whenever a process creates a file to determine its modes. The actual modes of the created file are the logical AND of the mask and the requested modes.

OPTIONS

+ Coupled with a r, w, or x adds that mode.

- Coupled with a r, w, or x subtracts that mode.

r,w,x Without + or - gives only that mode.

EXAMPLES

The following command line does not allow write access by the world to any files created after you have issued the command.

```
$setmask rwx rx
```

DIAGNOSTICS

none

ALSO SEE

pmask displays the current file creation mode mask

NOTES

setmask does not affect the modes of an already existing file.

setmask is a built-in command so that it affects the mask of the Command Interpreter and any commands invoked by it.

setmask

The default mask is: rwx rwx rwx

USES

nothing

setpri sets priority of a process

FORMAT

```
setpri [ceil=#][floor=#] pid1 pid2 ... pidn
```

DESCRIPTION

setpri sets the priority of a specified process. This command is a tool to help the system administrator manage the system by raising and lowering priorities on individual processes.

If you execute **setpri** without options, it displays the current process' priority, which is the priority of **setpri**.

Only ADMIN can raise the floor or ceiling.

OPTIONS

ceil=# Sets the ceiling priority of a process.

floor=# Sets the floor priority of a process.

EXAMPLES

To give the priority of the current process, type

```
$setpri
```

```
Process: 224 floor 0, ceil 100
```

To set the ceiling priority of a process to 50 and the floor to 10, type

```
$setpri ceil=50 floor=10 pid=134
```

DIAGNOSTICS

setpri checks the consistency of the request and if the request fails, makes its best guess as to the reason.

setpri: Can not read priorities. Access violation for process <number>/.

setpri

If someone other than the owner or the system administrator attempted to set the priority of a process.

ALSO SEE

nice executes a command at a given priority

NOTES

none

USES

nothing

setuser

setuser changes the current user and group ID

FORMAT

setuser [*user_name*]

DESCRIPTION

setuser changes your privileges to those of the specified user, or the system administrator, if no *user_name* is specified. You will be asked for the password, just as you were logging in as the specified user.

setuser changes the real and effective user IDs and group IDs (kept by the kernel for each process) and transfers control to the command processor. Any command you run at this level will have the access privileges of the new user ID, and group, and any files you create will be owned by the new user and group. However, you will still have your own abbreviations and home directory, and receive messages sent in your original name. When you exit from this command processor (by typing ^D or #q), you will return to your previous unprivileged state.

OPTIONS

none

EXAMPLES

To shutdown the system, which requires administrator privileges:

```
$setuser
```

Password:

```
#shutdown 0
```

```
LAST WARNING - SYSTEM SHUTTING DOWN IN 15 SECONDS  
SYSTEM SHUTTING DOWN NOW, GOODBYE
```

DIAGNOSTICS

```
setuser: Password Incorrect
```

setuser

ALSO SEE

nothing

NOTES

The command processor executed by **setuser** is determined by the seventh field of line in **/etc/passwd** which defines the new user.

USES

/etc/passwd To verify the password.

shutdown

shutdown brings the system to single user mode

FORMAT

shutdown minutes [reason]

DESCRIPTION

shutdown puts the system in single user mode. It signals the system control to terminate all extant processes. Only the system administrator, logged in on the system console can shutdown the system.

The number following the command determines when the **shutdown** occurs: 0 brings the system down in 15 seconds, 1 brings it down in 1 minute and 15 seconds etc.

If you give a reason, it will appear on all terminals with each warning message. The warning message is displayed every minute and then 15 seconds before the system shuts down.

shutdown kills all processes, except the system control process, and prints the message <**single user mode**> followed by the prompt # indicating that the system is now in single user mode. A ^D at this point will prompt the user with the message: **single, multi, or shutdown?** To shutdown the system, type **shutdown**, otherwise choose **single** or **multi**.

OPTIONS

none

EXAMPLES

To shutdown the sytem with a five minute warning and a reason, type:

```
$shutdown 5 for the evening
```

shutdown

DIAGNOSTICS

shutdown: Only the System Administrator can use the command

shutdown: You are not at the system console

ALSO SEE

UNOS System Concepts Manual
User's Guide to UNOS

NOTES

A ^C, up to the last possible moment, will abort the shutdown.
shutdown is moved to **/etc**.

USES

/etc/ttys finds out what terminals are on the system.
/dev/tty writes a message to each terminal.

size

size prints section sizes of common object files

FORMAT

size [-oxV] files

DESCRIPTION

size prints section size information for sections in COFF files. It prints sizes of the text, data, and bss sections and their 'sum. If *files* specified are archive files, **size** displays all the information for the archive members.

OPTIONS

- o Print numbers in octal instead of decimal, the default.
- x Print numbers in hexadecimal instead of decimal.
- V Print the current version information for the **size** command.

EXAMPLES

This example gets the section sizes of the **size** command.

```
shell> size /bin/size > on_a_hot_tin_roof
```

```
shell> cat on_a_hot_tin_roof
/bin/size: 23214 + 4194 + 4108 = 31516
```

DIAGNOSTICS

Self-explanatory

ALSO SEE

- cc** compiles C programs
- ld** links editor for common object files

size

NOTES

The `-v` option is not currently implemented.

USES

nothing

sleep

sleep delays execution for a specified interval

FORMAT

sleep time

DESCRIPTION

sleep pauses for <time> seconds. You can use it to execute a command after a certain amount of time has passed.

OPTIONS

none

EXAMPLES

105 seconds after you type the following command, the system will execute it in the background.

```
$(sleep 105; command) &
```

DIAGNOSTICS

none

ALSO SEE

nothing

NOTES

none

USES

nothing

space

space displays amount of free space in the file system

FORMAT

space [dev1...devn]

DESCRIPTION

space gives the amount of space (in blocks) that are free out of the total on each logical device specified.

If no files are given, each currently mounted partition is checked. Additionally, the directory on which each partition is mounted is printed.

OPTIONS

none

EXAMPLES

This command line prints the amount of space available on device xe4.

```
$space /dev/xe4  
/dev/rxe4: 2975 free out of 80000 blocks (high 78348)
```

To find the amount of space available on the CRYPT system type

```
$space /dev/rck?  
/dev/rck0:          25 free out of 2000 blocks (high 1974)  
/dev/rck2:      48224 free out of 72000 blocks (high 23903)  
/dev/rck3:       6042 free out of 72000 blocks (high 71672)  
/dev/rck4:     58920 free out of 60000 blocks (high 1079)  
/dev/rck5:      9325 free out of 20000 blocks (high 16486)  
/dev/rck6:      1202 free out of 2000 blocks (high 905)  
/dev/rck7:       25 free out of 2000 blocks (high 1974)
```

DIAGNOSTICS

```
<device>: bad file system header on <device>  
<device>: can't open <device>
```

space

ALSO SEE

diskusage counts blocks used by files

NOTES

The program does a sync to update the file system before it reads the allocation bit maps.

Using the *raw device* form is slightly faster.

USES

nothing

strip

strip strips symbol and line number information from a common object file

FORMAT

strip [-l] [-x] [-r] [-v] filename

DESCRIPTION

strip removes the symbol table and line number information from COFF files and archive files. You cannot use a symbolic debugger on a stripped file. Run **strip** only on debugged and tested production modules. **strip** reduces the file storage overhead taken by the object file. **strip** notifies you, then terminates if symbol table information is to be stripped and there are relocation entries in the object file. Use the **-r** flag in this case. If **strip** executes on a common archive file, it removes the archive symbol table. Restore the archive symbol table by executing the **ar** command with the **-s** option before **strip** executes on a link-edited by **ld**.

OPTIONS

Use these options to control the amount of information stripped from the symbol table:

- l** Strip line number information only; not symbol table information.
- x** Leave static or external symbol information.
- r** Reset the relocation indexes into the symbol table.
- V** Print the version of the strip command executing on the standard error output.

EXAMPLES

Strip can be used to reduce the size of an object file when debugging on that file has been completed.

```
$strip the_last_bug_is_out
```

strip

DIAGNOSTICS

Self-explanatory

ALSO SEE

ar archive and library manager for portable archives

cc compiles C programs

ld links editor for common object files

NOTES

none

USES

/usr/tmp/strp?????

su changes the current user id

FORMAT

su [-] [*user_name*] [*command*]

DESCRIPTION

su changes your privileges to those of the specified user, or the system administrator, if no *user_name* is specified. You will be asked for the password, just as if you were logging in as the specified user.

su changes the real and effective user IDs (kept by the kernel for each process) and transfers control to the command processor which is specified for the selected user in */etc/passwd*. Any command you run at this level will have the access privileges of the new user ID, and any files you create will be owned by the new user ID. However, you will still have your own abbreviations and home directory, and you will still receive messages sent to your original name. When you exit from this command processor (by typing **^D** or **#q**), you will return to your previous unprivileged state.

All arguments after *user_name*, if they exist, are taken to be a command line. The command is executed as though it had been typed to the command interpreter of the selected user.

OPTIONS

- causes the newly spawned shell to behave as a login shell, running initialization files and the like.

user_name user other than the system administrator.

EXAMPLES

To shutdown the system, which requires administrator privileges:

```
$su
```

Password:

SU

```
#shutdown 0
```

```
LAST WARNING - SYSTEM SHUTTING DOWN IN 15 SECONDS  
SYSTEM SHUTTING DOWN NOW, GOODBYE
```

To change the owner of the **ps** command to 'root', with one command line:

```
$su root chown root /bin/ps
```

Password:

```
$
```

DIAGNOSTICS

su: Password Incorrect

When the user's password does not match the one typed.

Exec of XXX failed

When su cannot execute the shell of the desired user. XXX refers to the shell name.

no such user

When user_name is not listed in /etc/passwd.

Bad flag: flag

Command line error.

ALSO SEE

newgrp enters a new group

setuser changes the current user and group ID

NOTES

none

USES

/etc/passwd To verify the password and select a command processor.

suname

suname sets system, node, and machine names

FORMAT

suname [-v] name node machine

DESCRIPTION

suname sets the system name, the node name, and the hardware name to *name*, *node*, and *machine* by invoking the **suname()** system call. If you omit the node name or the node and machine names, **suname()** is called using NULL for those corresponding arguments. You must be ADMIN in order for **suname** to set the system names.

OPTIONS

-v Sets the verbose flag. In verbose mode **suname** prints the **suname** subroutine call it makes to set the system names.

EXAMPLES

To set the system name to BONZO, the node name to WHOUSE, and the machine name to GADGETS, and to see the **suname** subroutine call that makes it all happen:

```
#suname -v BONZO WHOUSE GADGETS
suname(BONZO, WHOUSE, GADGETS)
```

DIAGNOSTICS

If an error occurs, **suname** outputs the error message:
Can't set system name

ALSO SEE

suname() sets name of current UNOS system

suname

NOTES

none

USES

nothing

suspend

suspend halts execution of a process

FORMAT

suspend pid1...pidn

DESCRIPTION

suspend halts the execution of the specified processes.

A process that has been suspended can either be continued with the **resume** command or terminated with the **kill** command.

OPTIONS

none

EXAMPLES

This command line suspends a background process.

```
$largeprogram &
```

```
123
```

```
..
```

```
..
```

```
..
```

```
$suspend 123
```

DIAGNOSTICS

suspend:

process <#> does not exist.

suspend:

access violation for process <#>.

ALSO SEE

kill sends a signal to processes

ps displays process status

resume resumes execution of a suspended process

suspend

NOTES

When you suspend a process that was run in the background, the Command Interpreter will inform you (the next time it waits for any command to complete) that the process is suspended with the message:

123: Suspended

USES

nothing

sync

sync updates disk with in-memory information

FORMAT

sync

DESCRIPTION

sync executes a system primitive, which causes all information in the system's disk buffers (cache) to be written out to the disk. You should use it before halting the system and before copying or dumping the contents of the disks.

OPTIONS

none

EXAMPLES

DIAGNOSTICS

none

ALSO SEE

nothing

NOTES

The **shutdown** and **unmount** commands automatically execute a **sync**.

In multiuser mode, the system runs the program **/etc/update** to execute a **sync** every 30 seconds.

USES

nothing

sysgen is a program which links system file with the specified input object files and updates table *devsw* of the linked file.

FORMAT

```
sysgen [o=name] [os=name] [l=name ... l=name] [device=#=name  
... device=#=name] file1 ... filen
```

DESCRIPTION

sysgen allows the user to add device drivers to the operating system. It uses the linker to combine the driver object file(s) with a kernel file, producing a new kernel file. It then modifies the device driver dispatch table (*devsw*) in the resulting kernel file so that the new driver(s) can be called. The index in the *devsw* table is the same number as the major device number specified to the *makedev* command when a device node for this driver is created.

OPTIONS

o=name places the output in the specified file instead of the default file */sys/unos.new*

os=name uses the file name for the kernel file instead of the default file */sys/unos*

l=name searches the file name as a library file

device=number=name replaces the entry number of the *devsw* with the address of the global name. For example, *device=5=dev* changes the fifth entry of *devsw* to the address of the global *__dev*

file1 ... filen names the input object files that will link with the kernel file

EXAMPLES

```
$sysgen o=unos.eagle eagle.j l=/lib/unos.lib  
device=13=eagle_table
```


sysgen

DIAGNOSTICS

Diagnostics display for

- bad options
- bad device table entry numbers
- link errors

ALSO SEE

link binds object modules

makedev creates a device file system entry

NOTES

none

USES

/bin/link

sysversion

sysversion displays UNOS version date

FORMAT

sysversion [sysfile ...]

DESCRIPTION

sysversion displays the version date of the currently running UNOS. If the pathnames of other versions of UNOS are given, the version date will be given for them also.

OPTIONS

None

EXAMPLES

```
$sysversion /unos.4.5
/unos.4.5: _Version: UNOS version Jun 5 1984 16:52
now running: UNOS version Jun 5 1984 16:52
```

This shows that the UNOS file named */unos.4.5* is the same version as the running UNOS.

DIAGNOSTICS

Self-explanatory.

ALSO SEE

nothing

NOTES

none

USES

nothing

tail

tail displays the first or last lines of a file

FORMAT

tail [-#][+#][-copy][file]

DESCRIPTION

tail prints on standard output the specified number of lines from the argument file, or the whole file if it contains fewer lines than the number specified. A negative argument displays the last lines in the file. A positive argument displays the first lines of the file. The default is the last 24 lines of the file. In the absence of a filename, **tail** reads from its standard input.

OPTIONS

-# Last # lines of a file.

+# First # lines of a file.

-copy Copies the file argument to a temporary holding file and uses the holding file to seek to the starting place.

EXAMPLES

To display the first 5 lines of the file called foo, type
\$tail +5 foo

DIAGNOSTICS

tail: File does not exist. Can not open 'filename'.

ALSO SEE

nothing

NOTES

tail determines the number of lines to print by first scanning back through the file instead of copying the file to a temporary

tail

file. It only makes a temporary copy of the file if the file is a device, standard in, or you specify the *-copy* option. Back scanning results in faster output.

The UNOS **tail** command has moved to **/cmd** because the UNIX command of the same name resides in **/bin**.

USES

/tmp/tl.* to hold the temporary copy.

ted creates and edits files

FORMAT

ted

[buffers=#][-nobak][-readonly][-tedhelp][term=name][file]

DESCRIPTION

ted is the UNOS screen-oriented editor. Type **ted** and specify the name of the file you want to edit. If you do not name a file, TED will assume the last file you edited in your current directory. When you edit a file for the first time always specify a file name.

OPTIONS

buffers=# Specifies the number of 1-Kbyte buffers used. If you do not use this option, TED uses 20 1-Kbyte buffers.

TED will page the file if it is larger than the allotted buffer space. This lets you edit large files without using huge amounts of memory.

-nobak Does not create a *.bak* file.

-readonly During the editing session, you can only read the files. This is the 'look-only' mode and since the file is not altered, the file's modification date remains unchanged.

-tedhelp The editor is called in readonly mode with the on-line help file.

term=name You can specify a terminal other than the one defined in your ENVIRONMENT variable. Check **/etc/termcap** for a list of possible *names*. This is a useful feature for times when you want to edit on a terminal other than the one you normally use.

ted

Command Format

To execute a TED command you press a combination of keys.

In the following discussion of TED commands, an up arrow (^) indicates the CTRL key and a dollar sign (\$) indicates the ESC key. All character keys are shown in uppercase. For instance, **\$^N** means press the ESC key, then press the CTRL key and hold it while you press the N key. (Note that this does not mean "capital n" or "shift n", just the "n" key.)

There are three types of TED commands:

1. **CTRL.** This type of TED command consists of a control character only. (A control character means that you press and hold the CTRL key while pressing another key. ^A, ^B, ^C, and so on.
2. **ESC CTRL.** This type of TED command consists of a control character preceded by the ESC key. \$^A, \$^B, \$^C, and so on.
3. **ALT.** These commands always begin with ^] -- a combination of keystrokes called ALT. The ASCII name for ALT is GS. ALT can precede both CTRL commands and ESC CTRL commands. ALT^A, ALT^B, ALT\$^C, and so on. Most ALT commands insert a mark in the text, usually at the current cursor position (see the individual TED command for details).

The function of all three types of commands can be altered by using a number argument. The number argument is always 1 unless you type \$#<cr> to change it (where # is the number you want the number argument to be). When you change the number argument, the next command you execute uses that number.

TED recognizes four character units: 9

1. **character**
a single letter, digit, symbol, space, or carriage return.
2. **word**
any sequence of characters terminated by a space, TAB, or carriage return.

3. line
any text terminated by a newline.
4. paragraph
any sequence of characters terminated by more than one
consecutive carriage returns.

In general, if a CTRL command moves the cursor by characters, preceding it with an ESC will move the cursor by words. If a CTRL command moves the cursor by lines, preceding it with an ESC will move the cursor by paragraphs. If a CTRL command moves the cursor in one direction, preceding it with an ESC will move the cursor in the opposite direction.

If you precede either a CTRL or an ESC command with ALT, a mark is inserted before the CTRL or ESC command function is performed.

Information Line

When you execute the **ted** command, the specified file is displayed on the screen. The top line on the screen contains information about the editing session. This first line is divided into the following five fields:

Field 1

Reserved for system messages, and questions. All confirmation is done here.

Field 2

Contains the current argument to the command. It is reset to 1 after every command and can be set by the user.

Field 3

The current buffer. Default is "default", however it can also be user set.

Field 4

The name of the file being edited.

Field 5

Error field. All warning messages appear here.

ted

The next page contains a single page summary of all TED commands. Then, the TED commands are discussed in detail, by functional category.

Command Summary

	CTRL	ESC CTRL
A	start of line	start of paragraph
B	beginning	bottom
C	exit	backup file
D	delete char	delete words
E	end of line	end of paragraph
F	forward char	forward word
G	get file	change file
H	back char	back word
I	tab	~
J	save char	save word
K	delete line	delete paragraph
L	adjust window	re-display
M	cr	change modes
N	down line	down paragraph
O	open line	~
P	up line	up paragraph
Q	XON	~
R	forward search	reverse search
S	XOFF	~
T	save line	save paragraph
U	#=mult	set mult
V	paste	paste and clear buffer
W	write file	write buffer
X	execute cmd	execute buffer
Y	suspend	help
Z	re-search	reversed
@	where from top	where from bottom
[	ESC	~
]	ALT	ALT ESC
\	change buffer	clr present buf
^	quote char	~
_	undo delete	~
DEL	erase character	erase word

ted

With a number argument, ^A moves the cursor that number of characters from the beginning of the current line.

\$^A moves the cursor to the beginning of the current paragraph. If the cursor is already at the beginning, it will not move.

With a number argument, \$^A moves the cursor that number of words from the beginning of the current paragraph.

ALT^A drops the mark at the current cursor position before moving the cursor to the beginning of the current line. If the cursor is already at the beginning, it will not move, but TED will still drop the mark.

With a number argument, ALT^A drops the mark, then moves the cursor that number of characters from the beginning of the current line.

ALT\$^A drops the mark at the current cursor position before moving the cursor to the beginning of the current paragraph. If the cursor is already at the beginning, it will not move, but TED will still drop the mark.

With a number argument, ALT\$^A drops the mark, then moves the cursor that number of words from the beginning of the current paragraph.

^B moves the cursor to the beginning of the file. If the cursor is already at the beginning of the file, it will not move.

With a number argument, ^B moves the cursor that number of lines from the beginning of the file.

\$^B moves the cursor to the end of the file. If the cursor is already at the end of the file, it will not move.

With a number argument, \$^B moves the cursor that number of lines from the end of the file.

ALT^B drops the mark at the current cursor position before moving the cursor to the beginning of the file. If the cursor is already at the beginning of the file, it will not move, but TED still drops the mark.

With a number argument, ALT^B drops the mark, then moves the cursor that number of lines from the beginning of the file.

ALT\$^B drops the mark at the current cursor position before moving the cursor to the end of the file. If the cursor is already at the end of the file, it will not move, but TED still drops the mark.

With a number argument, ALT\$^B drops the mark, then moves the cursor that number of lines from the end of the file.

^E moves the cursor to the end of the current line. If the cursor is already at the end of the line, it will not move. The last character on the line is the invisible ASCII newline character NL; when the cursor is at the end of the line, it is not on a visible character.

With a number argument, ^E moves the cursor that number of characters from the end of the current line.

\$^E moves the cursor to the end of the current paragraph. If the cursor is already at the end, it will not move.

With a number argument, \$^E moves the cursor that number of words from the end of the current paragraph.

ALT^E drops the mark at the current cursor position before moving the cursor to the end of the current line. If the cursor is already at the end of the line, it will not move, but TED still drops the mark.

With a number argument, ALT^E drops the mark, then moves the cursor that many characters from the end of the current line. For instance, if the cursor is at the beginning of the line and the number argument is 10, the cursor

ted

moves to the tenth character from the end of the line while the mark is dropped at the beginning of the line.

ALT\$^E drops the mark at the current cursor position before moving the cursor to the end of the current paragraph. If the cursor is already at the end of the paragraph, it will not move, but TED still drops the mark.

With a number argument, ALT\$^E drops the mark, then moves the cursor that number of words from the end of the current paragraph. For instance, if the cursor is at the beginning of the paragraph and the number argument is 10, the cursor moves to the tenth word from the end of the paragraph while the mark is dropped at the beginning.

^F moves the cursor one character position to the right.

With a number argument, ^F moves the cursor that many characters to the right.

Depending on the cursor position, with or without a number argument, **^F** could move the cursor down to the next line, or further. If the cursor is at the end of the line, the cursor will move to the first position of the next line.

\$^F moves the cursor to the first letter of the next word.

With a number argument, \$^F moves the cursor that many words forward.

Depending on the current cursor position and the number argument, **\$^F** might move the cursor down one or more lines.

ALT^F drops the mark at the current cursor position before moving the cursor one character to the right.

With a number argument, ALT^F drops the mark, then moves the cursor that many characters to the right.

ALT\$^F drops the mark at the current cursor position before moving the cursor to the first letter of the next word.

With a number argument, **ALT\$^F** drops the mark, then moves the cursor that many words to the right.

^H moves the cursor one character position to the left.

With a number argument, **^H** moves the cursor that many characters to the left.

Depending on the cursor position, **^H**, with or without a number argument, could move the cursor up to the previous line. If the cursor is at the beginning of the line when you press **^H**, the cursor will move to the last position of the previous line.

\$^H moves the cursor to the first letter of the word just prior to its current position.

With a number argument, **\$^H** moves the cursor that many words backwards.

Depending on the current cursor position and the number argument, the cursor may move up one or more lines.

ALT^H drops the mark at the current cursor position before moving the cursor one character position to the left.

With a number argument, **ALT^H** drops the mark, then moves the cursor that number of characters to the left.

ALT\$^H drops the mark at the current cursor position before moving the cursor to the first letter of the previous word.

With a number argument, **ALT\$^H** drops the mark, then moves the cursor that many words to the left.

^N moves the cursor down one line while maintaining its horizontal position. If the cursor is on the last character of the last line when you press **^N**, the cursor will not move and you will hear a beep.

With a number argument, **^N** moves the cursor down that number of lines.

ted

$\N moves the cursor to the beginning of the next paragraph.

With a number argument, $\N moves the cursor down that number of paragraphs.

ALT^N drops the mark at the current cursor position before moving the cursor down one line. Note that the cursor maintains its horizontal position on the line (if possible -- differing line lengths may prevent TED from maintaining the horizontal position); i.e., if the cursor was on the fifth character of the line when you pressed **ALT^N** , it will move to the fifth character of the line below (if the next line has that many characters).

With a number argument, ALT^N drops the mark, then moves the cursor down that number of lines.

$ALT\N drops the mark at the current cursor position before moving the cursor to the beginning of the next paragraph.

With a number argument, $ALT\N first drops the mark. Then it moves the cursor down that number of paragraphs.

P moves the cursor up one line while maintaining its horizontal position, if possible. If the cursor is on the first line of the file when you press **P** , the cursor will not move and you will hear a beep.

With a number argument, P moves the cursor up that number of lines. If the number argument is greater than the number of lines between the cursor and the beginning of the file, the cursor will stop moving at the beginning of the file.

$\P moves the cursor to the beginning of the previous paragraph.

With a number argument, $\P moves the cursor that number of paragraphs prior to the cursor's position.

ALT^P drops the mark at the current cursor position before moving the cursor up one line. Note that the cursor maintains

its horizontal position on the line (if possible -- differing line lengths may prevent TED from maintaining the horizontal position).

With a number argument, ALT^P drops the mark, then moves the cursor up that number of lines.

ALT\$^P drops the mark at the current cursor position before moving the cursor to the beginning of the previous paragraph.

With a number argument, ALT\$^P first drops the mark. Then it moves the cursor up that number of paragraphs prior to its current position.

Addition/Deletion

^D erases the character at the cursor position.

With a number argument, ^D erases that many characters to the right of the cursor.

\$^D erases the next word. A word is defined from the current cursor position to the next blank or carriage return.

With a number argument, \$^D erases that number of words, to the right of the cursor.

^I inserts a tab. The cursor moves to the next tab stop when the line is blank. If you use **^I** on a line that contains text, the text to the right of the cursor also moves over one tab stop. Tab stops are every eight characters on the screen.

With a number argument, ^I inserts that number of tabs.

ALT^I drops the mark at the current cursor position before inserting tabs.

With a number argument, ALT^I drops the mark, then inserts that number of tabs.

ted

^K erases the current line. If the cursor is located at the beginning of the line, the entire line is erased, including the carriage return at the end of the line. If the cursor is located between the beginning and end of the line, everything right of the cursor is erased, including the carriage return at the end of the line. You can use **^K** to erase blank lines.

With a number argument, ^K erases that number of lines from the current cursor position.

\$^K erases from the current cursor position to the end of the paragraph. The end of a paragraph is indicated by two successive carriage returns.

With a number argument, \$^K erases that number of paragraphs.

ALT^K erases text between the mark and the current cursor position.

^M inserts a carriage return at the current cursor position and moves the cursor to the beginning of the next line.

With a number argument, ^M inserts that number of carriage returns.

^O inserts a newline at the current cursor position, but does not move the cursor. Any text to the right of the cursor moves down to start a new line.

With a number argument, ^O inserts that number of carriage returns without moving the cursor.

ALT^O inserts a carriage return without moving the cursor. It drops the mark at the beginning of the next (new) line.

With a number argument, ALT^O inserts that number of carriage returns without moving the cursor. It drops the mark after the last carriage return.

^DELETE erases the character to the left of the cursor. This acts the same as using the DELETE key alone.

With a number argument, ^DELETE erases that number of characters to the left of the cursor.

\$^DELETE erases the word to the left of the cursor. This acts the same as using **\$DELETE**.

With a number argument, \$^DELETE erases that number of words to the left of the cursor.

^_ reinserts the last deletion.

ALT^_ erases the mark.

Cut/Paste

These commands deal with buffers. You can have up to 20 buffers, each named as you choose. The default buffer is "default".

^J removes a character from the file and puts it in the current buffer. The character at the current cursor position is the one taken.

With a number argument, ^J takes that number of characters and puts them in the current buffer.

\$^J removes a word from the file and puts it in the current buffer. A word consists of any characters from the current cursor position up to the next blank or carriage return.

With a number argument, \$^J takes that number of words and puts them in the current buffer.

^T removes a line from the file and puts it in the current buffer. The line is defined from the current cursor position to the next carriage return.

With a number argument, ^T removes that number of lines from the file and puts them in the current buffer.

ted

\$^T removes a paragraph from the file and puts it in the current buffer. A paragraph is defined from the current cursor position to two successive carriage returns -- one at the end of the last line and one on the blank line that follows the paragraph.

With a number argument, **\$^T** removes that number of paragraphs from the file and puts them in the current buffer.

ALT^T removes the selection (what lies between the mark and the cursor) and puts it in the current buffer. (This is the same as **ALT\$^T**.)

ALT\$^T removes the selection (what lies between the mark and the cursor) and puts it in the current buffer. (This is the same as **ALT^T**.)

^V pastes the contents of the current buffer into the file at the current cursor position. The buffer will still contain a copy of the text.

With a number argument, **^V** pastes the contents of the current buffer that many times. The buffer will still contain a copy of the text.

\$^V pastes and clears the contents of the current buffer into the file at the current cursor position. The buffer will be empty.

With a number argument, **\$^V** pastes the contents of the buffer that many times. The contents are also cleared out of the buffer so that it will be empty.

ALT^V replaces the selection (what lies between the mark and the cursor) with whatever is in the current buffer. The buffer will still contain a copy of the text.

ALT\$^V replaces the selection (what lies between the mark and the cursor) with whatever is in the current buffer and clears the buffer. The buffer will be empty.

^_ 3 changes the current buffer.

On the information line you will see the **Take Buffer:** prompt. Type a buffer name and press the RETURN key.

If you type a new name, a new buffer is created. If you type a name you have already used during the current editing session, you can reuse that buffer. The buffer name is always displayed on the information line.

\$^_ clears any text out of the current buffer. On the information line you will see the **Buffer cleared** message.

Search

^R searches forward in a file for the specified text string.

On the information line you will see the **+Search:** prompt (the + indicates a forward search). Type the string you want to find. If the string exists between the cursor's current position and the end of the file, TED moves the cursor to the end of the matched string. If TED can not find the specified string, the cursor does not move and you will hear a beep.

With a number argument, ^R finds that numbered occurrence of the string. For instance, if the number argument is 2, the cursor moves to the second occurrence of the string, skipping over the first one.

\$^R searches backwards in a file for the specified text string.

On the information line you will see the **-Search:** prompt (the - indicates a backward search). Type the string you want to find. If the string exists between the cursor's current position and the beginning of the file, TED moves the cursor to the end of the matched string. If TED can not find the string, the cursor does not move and you will hear a beep.

With a number argument, \$^R finds that numbered occurrence of the string. For instance, if the number

ted

argument is 3, the cursor moves to the third occurrence of the string, skipping over the first two.

ALT^R searches forward in a file for the specified text string.

On the information line you will see the **+Search:** prompt (the + indicates a forward search). Type the string you want to find. When (and if) found, the mark is dropped at the beginning of the string and the cursor moves to the end of the string; the string is then considered selected. If TED can not find the string, the cursor does not move, no mark is dropped, and you will hear a beep.

With a number argument, ALT^R finds that numbered occurrence of the string. For instance, if the number argument is 4, the mark is dropped at the beginning of the fourth occurrence of the string.

ALT\$^R searches backwards in a file for the specified text string.

On the information line you will see the **-Search:** prompt (the - indicates a backward search). Type the string you want to find. When found, the mark is dropped at the beginning of the search string and the cursor moves to the end of the string; the string is then considered selected. If TED can not find the string, the cursor does not move, no mark is dropped, and you will hear a beep.

With a number argument, ALT\$^R finds that numbered occurrence of the string. For instance, if the number argument is 3, the mark is dropped at the beginning of the third occurrence of the string.

^Z re-searches for the last text string you asked for with the **^R**, **\$^R**, **ALT^R**, or **ALT\$^R** command. If you already found one occurrence of the text string, ^Z will find the next occurrence. If TED can not find another occurrence, the cursor does not move and you will hear a beep.

With a number argument, ^Z finds that numbered occurrence of the string. For instance, if the number

argument is 2, the cursor moves to the second occurrence of the string, skipping the next one.

\$^Z reverses direction and re-searches for the last text string you asked for with the **^R**, **\$^R**, **ALT^R**, or **ALT\$^R** command. If you already found one occurrence of the text string, **^Z** will find the next, or previous, occurrence, depending on which your direction. For instance, if the original command was **^R**, **\$^Z** will find the previous occurrence. If TED can not find another occurrence, the cursor does not move and you will hear a beep.

With a number argument, \$^Z finds that numbered occurrence of the string. For instance, if the number argument is 2, the cursor moves to the second occurrence of the string, skipping the next one.

ALT^Z re-searches for the last text string you asked for with the **^R**, **\$^R**, **ALT^R**, or **ALT\$^R** command. If you already found one occurrence of the text string, **ALT^Z** will find the next occurrence, drop the mark at the beginning of it, and move the cursor to the end of it. This occurrence of the string is then considered selected. If TED can not find another occurrence, the cursor does not move, no mark is dropped, and you will hear a beep.

With a number argument, ALT^Z finds that numbered occurrence of the search string, with the numbering beginning from the current cursor position. For instance, if the number argument is 4, the next three occurrences are skipped and the mark is dropped at the beginning of the fourth occurrence of the string. This happens regardless of how many occurrences of the string have already been located, unless there are not that many occurrences of the string in the file.

ALT\$^Z reverses direction and re-searches for the last text string you asked for with the **^R**, **\$^R**, **ALT^R**, or **ALT\$^R** command. If you already found one occurrence of the text string, **ALT\$^Z** will find the next, or previous, occurrence, depending on your direction. For instance, if the original command was **^R**, **\$^Z** will find the previous occurrence,

ted

drop the mark at the beginning of it, and move the cursor to the end of it. This occurrence of the search string is then considered selected.

With a number argument, ALT\$^Z finds that numbered occurrence of the search string, with the numbering beginning from the current cursor position. For instance, if the number argument is 4, the next 3 occurrences of the string are skipped and the mark is dropped at the beginning of the fourth occurrence. This happens regardless of how many occurrences of this string have already been located, unless there are not that many occurrences of the string in the file.

Exiting/Putting

^C ends the editing session.

On the information line you will see the **QUITTING. PUT EDITS? (Y/N)** prompt. Type Y (and press the RETURN key) if you want to save the changes made during this editing session. Type N (and press the RETURN key) if you don't want to save the changes. If don't want to end the session, type anything except Y or N.

The current backup file (if there is one) (called *filename.bak*) is deleted, the current file is renamed *filename.bak*, and a version of the file containing the latest changes becomes *filename*.

\$^C backs up the current file.

The current backup file (if there is one) (called *filename.bak*) is deleted, the current file is renamed *filename.bak*, and a version of the file containing the latest changes becomes **filename**.

^Y suspends the current editing session.

On the information line you will see the **SUSPENDING. PUT EDITS? (Y/N)** prompt. If you want to file updated, type Y and press the RETURN key, if you do not, type N and

press the RETURN key.

When you are ready to continue editing, use the UNOS **resume** command. You will return to the editing session. The cursor will be in the same place it was when you suspended the session, regardless of whether or not you saved your edits.

Files

^G lets you insert another file at the current cursor position.

On the information line you will see the **Get from:** prompt. Type a file name and press the RETURN key.

You can insert files from the current working directory or from other directories (as long as you use a valid pathname).

\$^G exits the current file and enters another file.

On the information line you will see the **Change to:** prompt. Type the name of the file you wish to edit and press the RETURN key. Then you will see the **CHANGING. PUT EDITS? (Y/N)** prompt; if you want to save the edits in the current file, type Y (and press the RETURN key), if you do not, type N (and press the RETURN key).

^W writes a copy of the file you are editing to the file you specify.

On the information line you will see the **Write to:** prompt. Type a file name and press the RETURN key. If the file exists, its contents are overwritten. If the file does not exist, it is created.

\$^W writes the contents of the current buffer to the file you specify.

On the information line you will see the **\ to file:** prompt. Type a file name and press the RETURN key. If the file

ted

exists, its contents are overwritten. If the file does not exist, it is created. The buffer contents will remain intact.

ALT^W writes a copy of the selection (what lies between the mark and the cursor) to the file you specify.

On the information line you will see the **Sel to:** prompt. Type a file name and press the RETURN key. If the file exists, its contents are overwritten. If the file does not exist, it is created.

Numbers

^U multiplies the number argument by the multiplier. The default multiplier is 4. Each time you press **^U** the number argument changes to the next multiple of 4 (16, 16, 64, upto 4194304 -- after that you'll see **BIG, BIG, BIG,** and **BIG,** and then the error message **Number Overflow!;** the argument then returns to 1).

\$^U changes the multiplier.

The default multiplier is 4. If you want to multiply number arguments by a different number, use **\$^U**. On the information line you will see the **Mult =** prompt. Type a number and press the RETURN key.

Miscellaneous

ALT^G lets you edit your macro file. It is the same as if you used **^X** to execute a new TED and answered **N** to the **Put** question. When you are through editing **.EDMAC**, press **^C** to continue editing the file you were in when you issued the **ALT^G** command; you will be asked whether or not you want to put the edits from **.EDMAC** before you return to the original file.

Note that after using the **ALT^G** command, the **.EDTMP** file will read **.EDMAC**. This means the next time you try to edit a file in this directory by executing the **ted** command with a file name, the **.EDMAC** file is the one you'll enter.

ALT^J moves the cursor to the mark.

The current cursor position determines whether this is a forward or backward movement. For instance, if the mark is in the middle of the file and the cursor is at the beginning, its a forward movement; if the cursor is at the end, its a backward movement.

ALT\$^J switches the cursor and the mark.

^L adjusts the text displayed on the terminal screen so that the line the cursor is on becomes the sixth from the top. This may involve displaying different text, depending on where the cursor is when you use this command.

\$^L redisplayes the current screenful of text. This is useful after a message has been sent to your terminal which may have overwritten some of the text.

\$^M changes modes.

On the information line you will see the **Mode?(ORV)** prompt. Type O for overstrike mode; type R to reset back to default mode; type V for visible mode. If you type anything else, you will abort the command.

ALT^M puts a mark at the current cursor position.

^Q unfreezes the screen which was frozen by pressing **^S**. This is not a TED command; it is a function of the tty driver.

^S freezes cursor and text movement on the screen. Use **^Q** to unfreeze it. This is not a TED command; it is a function of the tty driver.

^X lets you execute a UNOS command without terminating the editing session.

Control temporarily returns to the Command Interpreter while the command is executed, then you can continue editing. When you type **^X**, you will see the **Execute:** prompt on the information line. Type the command line and

ted

press the RETURN key. Then you will see the **EXECUTING. PUT EDITS? (Y/N)** prompt. If you type **Y**, the file is updated and when you return to the editing session, the cursor will be at the beginning of the file. If you type **N**, the file will not be updated and when you return to the editing session, the cursor will be in the same place it was when you used the **^X** command.

You can redirect the input or output of the command you are executing to the current buffer by using the backslash (****). For instance, if you want to insert the results of a **list** command into the file you're editing, use the **^X** command and then execute the following command line:

```
list > \
```

The result of the **list** command will be in the current buffer.

Note that if the UNOS command you want to execute requires the current file as an argument, you might need to update it. For instance, if you the file you are editing contains formatting commands that you want to test, you will need to update the file so that the version of the file you format and view is the most up-to-date one.

To abort the command while you can see the **Execute:** prompt, do not type a command; simply press the RETURN key. If you had started to type a command, use the DELETE key to erase it; the field following the prompt must be blank. To abort the command while you can see the **EXECUTING. PUT EDITS? (Y/N)** prompt, type anything except a **Y** or **N**.

\$^X executes the contents of the current buffer.

You can type a UNOS command line in your file and then save it in a buffer (using the **^T**, **\$^T**, **^J** or **\$^J** commands). When you type **\$^X**, you will see the **EXECUTING. PUT EDITS? (Y/N)** prompt on the information line. If you want the file updated, type **Y** and press the RETURN key, if you do not, type **N** and press the RETURN key.

ALT^X executes the selection (what lies between the mark and the cursor).

You can type a UNOS command line in your file and select it. When you type **ALT^X**, you will see the **EXECUTING. PUT EDITS? (Y/N)** prompt on the information line. If you want the file updated, type Y and press the RETURN key, if you do not type N and press the RETURN key.

To abort the command, type anything except a Y or N.

\$^Y displays the on-line help file.

On the information line you will see the **HELP(Y?)** prompt. To continue editing (and abort this command), type N. To see the help file, type Y.

^@ displays the current line number. This number tells you how many lines exist between the beginning of the file and the current cursor position.

Note that this command is terminal dependent; it may not work on all terminals.

\$^@ displays the number of lines that exist between the current cursor position and the end of the file.

Note that this command is terminal dependent; it may not work on all terminals.

ALT^@ displays the number of the line that the mark is on, in relation to the beginning of the file.

If there is no mark, you will see **No mark!** displayed on the far right of the information line.

Note that this command is terminal dependent; it may not work on all terminals.

ALT\$^@ displays the number of the line that the mark is on, in relation to the end of the file.

ted

If there is no mark, you will see **No mark!** displayed on the far right of the information line.

Note that this command is terminal dependent; it may not work on all terminals.

^[is the same as using the **ESC** key.

^] is **ALT**. Use this sequence of keystrokes to start an **ALT** command.

\$^] is the same as using **ALT\$**.

^^ quotes the next character so that it is taken literally.

This command is terminal dependent; it may not work on all terminals.

Note that this command is control-uparrow; press the **CTRL** key and hold it while you press the uparrow (shift-6 on most terminals) key.

Macros

You can combine different TED commands to create your own commands or macros. To create macros edit the **.EDMAC** file in your home directory. You can edit it as a regular file or with the special TED command, **ALT^G**. Each macro definition has three fields: the macro name, the macro's function, and any comments. Separate fields one and two with colons and field three with a carriage return. For example the following macro, **bp** moves the cursor to the beginning of the paragraph:

bp:^[^A:begin paragraph

To execute a macro press the **ESC** key and type the macro name. You do not need to delimit the command. The **ESC** is not part of the macro name. If TED can not find the macro, you will hear a beep.

If your terminal has programmable function keys, you can place macros in them. Then to execute the macro, you need only press the key.

You can not combine macros when you execute them; nor can you execute a macro that depends on a previously executed command. You can, however, use number arguments with macros. In the above macro for instance, you could move to the third or fourth paragraph.

EXAMPLES

none

DIAGNOSTICS

none

ALSO SEE

UNOS Text Processing
UNOS System Concepts Manual

NOTES

If the eighth bit of a character is "on", the tilde character ~ appears before characters, i.e., ~C.

USES

p/tedB.pid# Backwards delete take buffer.
/tmp/tedD.pid# Delete buffer.
/tmp/tedF.pid# Forward delete character.
/tmp/tedT.pid Take buffer.
/tmp/tedX.pid# Command buffer.
/tmp/ted.default.pid# Default buffer.
/etc/termcap For terminal descriptors.
/doc/cmds/ted.help On-line help file.
.EDTMP Stores the name of the last file edited.

tee duplicates output in files

FORMAT

```
tee [-a][-err][-flush][-i] file1...filen
```

DESCRIPTION

tee takes its standard input and creates a T-type plumbers joint in the stream. One branch appears on the standard output and the other branch is written to each and every specified file. It is a way of both monitoring and catching output from a program or saving intermediate data in a pipe.

OPTIONS

-a Appends to named files, rather than truncating them.

-err Output to standard error, standard out, and specified files.

-flush No internal buffering. Output is a character at a time.

-i Ignore signals -- SIGINT (^C) and SIGQUIT (^Y).

EXAMPLES

```
$list -l | tee listing
```

The result of **list** appears on the screen and also in the file **listing**.

```
$cut -d: -f3,1 /etc/passwd |tee people | quicksort -n
```

cut divides the *passwd* file into the specified sections, puts the results into the *people* file, and uses it as input to the **quicksort** command. The final output is displayed on the screen.

This command line gets a log of the output of a **debug** session:

```
$debug | tee -flush log
```

```
$format chap.1 | tee -err | lpr
```

This command line puts the output of the **format** on the screen as it is being sent to the line printer.

tee

DIAGNOSTICS

none

ALSO SEE

User's Guide to UNOS

NOTES

none

USES

nothing

term asynchronous terminal emulator

FORMAT

```
term host_connect_port -nolock  
    [ftrans <file name>]  
    [frecv <file name>]
```

DESCRIPTION

term allows a Charles River terminal user to connect to a remote system via asynchronous lines and emulate the functions of a "dumb" terminal. It supports connection via async lines and modems.

For example, you can use this facility to access public data bases such as DOW JONES or COMPUSERV or other Charles River systems as a terminal. Data can be stored in local files or sent from local files.

term also allows a Charles River terminal user to transfer files.

The asynchronous terminal emulator capability (ASTECC) lets Charles River system users access other Charles River and non-Charles River systems.

Two optional commands, **ftrans** and **frecv**, are used in conjunction with **term**; **term** must be executed first, before either **ftrans** or **frecv** can be executed.

ftrans is the remote interface between **term** and the receiver.

frecv is the remote interface between **term** and the transmitter.

Before executing **term**, consider the following:

- baud rates and other characteristics should be set for host connect port using a command like **ttymodes**.
- the baud rate of the user terminal connection should be greater than or equal to the host connect baud rate.

term

- terminal type coordination with the host must be present just as when connecting a really dumb terminal.

It is possible to invoke the following functions after **term** is running

- ~s ^C send using input from a file (file name prompted).
- ~l ^C execute commands locally. **TERM LOCAL:** is the prompt. Type the word reset to exit local mode.
- ~d ^C place screen output into a file (name prompted).
- ~r ^C reset ~send and ~recv streams to default.
- ~. ^C end the **term** session.

file transfer described below.
Note that "~^C" means "control c".

File Transfer

Retrieving a remote file

A file transfer can be initiated after **term** has logged into another CRDS machine by typing in the command:
\$ftrans <file name>

This command line initiates a transfer from the remote machine to the local machine to transfer the named file. The file is copied to the current working directory after appending **.remote** to its name.

A log of the transfer is kept in **/tmp/_fnet_log** on the remote machine.

Send a local file to a remote system

frecv sends a local file to a remote CRDS system. The file will be copied to the users current working directory after appending the user's logon name. The syntax for **frecv** is
\$frecv <file name>

Logging information is kept the same as in **ftrans**.

OPTIONS

fttrans optional command for remote interface between **term** and the receiver.

frecv optional command for remote interface between **term** and the transmitter.

-nolock Normally the connection port is locked (**filelock()**). The *-nolock* option won't lock the port. Be careful.

EXAMPLES

To use **term**:

```
$term /dev/tty7
```

where **/dev/tty7** is the host connect port.

Once **term** is executed, anything that is typed will be sent out over the host connect port. In addition, anything which is sent from the host will be displayed on the terminal screen.

For instance, to log onto a remote Charles River system, type

```
$term /dev/tty7
```

Then, to retrieve a remote file, type

```
$fttrans test.file
```

To end the **term** session, type

```
$~.^C
```

DIAGNOSTICS

EEK! Buffer overflow! Guaranteed character loss! Use lower baud rate!

Caused by heavy local (host) machine usage, presumably by other users, or high connection baud rates, generally in excess of 9600 baud. Lowering the baud rate between machines usually eliminates this problem.

term

ALSO SEE

ttymodes sets or displays terminal modes

NOTES

All ASCII characters can be sent to the remote machine. "Break" cannot be sent.

To send the "~" (tilde) character, type two of them -- "~~".

^S and **^Q** are used to stop and restart sending.

Direct connection to other systems can be done at very high speeds, but keep in mind that no error correction and detection beyond parity is being performed by **term**. If a file transfer is being performed, then error correction is supported.

This facility is intended for lower speed, "casual" data inquiries. The optional commands must be used after **term** is invoked. Execution of **frexv** or **ftrans** without first invoking **term** on the first machine will produce no useful results.

USES

nothing

time

time displays execution time for a command

FORMAT

time command

DESCRIPTION

time prints how long it took, in seconds and tenths of seconds, to execute a command. After the specified command completes, **time** prints time spent in system mode, time spent in user mode, and elapsed real time.

The times are printed on the standard error stream.

OPTIONS

none

EXAMPLES

```
$time list  
system: 2.6  
user:   2.1  
real:  14.2
```

DIAGNOSTICS

time reports execution failures.

ALSO SEE

nothing

NOTES

time does not take into account the time used to begin execution of the **time** command itself.

time

time does not use the Command Interpreter to execute the specified command. Therefore, you can not use "begin" abbreviations (#b) with the **time** command.

The UNOS **time** command has moved to **/cmd** because the UNIX command of the same name resides in **/bin**.

USES

nothing

tp performs magnetic tape operations

FORMAT

tp [-flags] [option=value] [filenames]

DESCRIPTION

tp performs positioning and data transfer operations on a magnetic tape unit. **tp** is designed to keep the 1/2" tape drive streaming continuously for maximum speed, but it can also has limited use with the 1/4" tape drive.

tp transfers the named files to or from tape. Each file becomes a separate file on the tape, separated by the standard tape mark (file mark). There is no label or header information, so the file has no name on the tape.

tp rewinds by default. The *-nocrewind* option overrides this default. If you do not rewind the tape, any action on the tape is relative to the tape's current position.

Nothing except the rewind happens unless it is specifically requested by one of the options below.

OPTIONS

Most options which accept a number allow the number to be followed by the letters *b*, *k*, or *m*, meaning multiply the number by 512, 1024, or 1024*1024, respectively.

Data Movement Options

-p or **-put** Transfers a file or files to the tape. If no files are specified, **stdin** is used.

-g or **-get** Transfers a file or files from the tape. If no files are specified, **stdout** is used.

-v or **-verify** Verifies that a disk file and a tape file are the same. If no files are specified, **stdin** is used.

tp

dskip=# Specifies the number of bytes to skip on each of the named (disk) files.

bs=# Sets the size of the buffers used to copy data from or to the disk and to or from the tape. This becomes the size of the data blocks written on the tape, unless overridden by the *setbs*=# option, except that the last block of each file may be shorter (but see *-fill*). Large values of *bs* may prevent the tape from streaming (moving continuously) because the disk transfer then takes more time than the tape can run unattended.

bs Defaults to *setbs* (if specified) or to 8192.

-fill Forces all tape writes to be full length (*bs*), by padding the buffer with zeros.

count=# Specifies the number of disk buffer-loads copied to tape. Ignored during transfer from tape to disk. By default, all of the file is transferred.

-s or *-silent* Turns off printing of file movement information.

-verbose Displays verifying output, such as transfer size, and time taken by each part of transfer.

You can combine *-put*, *-get*, and *-verify* options. Puts happen first, then gets, and finally verifies, regardless of the order of the options on the command line.

Positioning Options

Actions requested by these options occur in the order in which they are listed here.

-noc or *-nocrewind* Does not rewind the tape, either before or after processing.

-no or *-norewind* Does not rewind the tape initially. **tp** may still rewind the tape after processing if *-nocrewind* is not specified.

- erase* Erases the entire tape from the current position to the physical end, then rewinds.
- append* Appends to the end of the tape. (Moves forward to two adjacent tape marks and then backspaces one tape mark.)
- tskip=#* Specifies a number of files to skip on the tape before processing. The number may be negative, meaning move backwards over some files.
- nc* or -*ncrewind* Does not rewind the tape after processing. *tp* may still rewind the tape initially if -*norewind* is not specified.
- un* or -*unload* Rewinds and unloads the tape after processing.

Tape Specific Options

tape=tapename Sets the name of the tape device for **tp**.

By default, **tp** tries these names in sequence:

/dev/rmt/0m.raw	1/2" tape on SASI bus
/dev/tm.async	1/2" tape on Tape Master
/dev/rct0.raw	1/4" tape on SASI bus
/dev/rcs0.raw	ditto, another controller

- con* or -*conserve* Prevents the inter-record gap from increasing during writes. This option may keep the tape from moving continuously, which makes the write slower.
- slow* Runs the tape at reduced speed. This option may result in higher throughput if the tape cannot be kept in continuous motion at full speed. With the SASI 1/2" tape, this option also forces the controller to write to the tape immediately (rather than accumulating data in a writebehind buffer).
- density=#* Specifies the data recording density on the tape. Valid densities are 1600, 3200, and 6250 (bytes per inch, bpi), although each drive supports only some pair of these densities. The default density is 1600 bpi.

tp

setbs=# Sets the size of the blocks that **tp** reads or writes on the tape, which can be any divisor of the buffer size (set by *bs=#*). This option pads all tape writes to the next multiple of *setbs*.

Sets the tape block size used by the tape driver when translating seek calls into tape motion. For instance, if you have a tape blocked at 2000 bytes per block and a program that must perform a seek before reading the tape, use a **tp setbs=2000** to set the block size before invoking your program.

pbuf=# Specifies the maximum number of physical transfers that you can schedule. The default is 5 for the Tape Master controller, and 1 for everything else.

abuf=# Specifies total number of buffers. When dealing with large block sizes on small memory systems, set *abuf* to 1. Default is *pbuf*.

-noerror Continues after an error.

EXAMPLES

This example copies all the files in a directory to a tape. Note that a subdirectory within the current directory copies onto the tape, but the files contained within the subdirectory do not copy.

```
$tp -p *
```

This example copies the first 10 files from a 3200 bpi tape to files a1 through a10.

```
$tp -g density=3200 a1 a2 a3 a4 a4 a6 a7 a8 a9 a10
```

This example rewinds the tape, copies the 3 Winchester partitions to tape using 10240 byte records, verifies that the tape and Winchester match, and rewinds the tape.

```
$tp -p -v /dev/rcd0 /dev/rcd3 /dev/rcd4 dskip=6b bs=10k
```

NOTE: Do not mount the file system when transferring its contents to and from a tape.

This example restores the 3 Winchester partitions from the tape. The *dskip* option allows doing this without overwriting certain per-disk data that may occupy the first 6 blocks of */dev/rcd0*, but the tape must have been created with the same *dskip* value.

```
$tp -g /dev/rcd0 /dev/rcd3 /dev/rcd4 dskip=6b bs=10k
```

This example copies a filesystem to tape in 512-byte tape records. This method wastes tape because there is a .6" gap after each record, but you can actually mount (*-readonly*) the resulting tape file, in case you want to list or extract a few files without the bother of copying the whole file system back to a free partition on disk.

```
$tp -p /dev/rcd0 setbs=1b
```

To use this example with use the Tape Master controller, is very slow. The driver does not allow writing multiple tape blocks from a single buffer. Reading a disk in such small pieces is so slow that the disk can only keep up with the tape at its slower speed. *-conserve* prevents wasting tape in case the disk temporarily falls behind.

```
$tp -p /dev/rcd0 bs=1b -slow -conserve
```

This example rewinds and erases the tape and copies **stdin** (the output of *some_command*) to the tape. Rewinds the tape. The drive does not require erasing before writing, but erasing can prevent accidentally reading old data. Suppose *some_command* runs all day, only occasionally emitting an output record. If the system crashes, there won't be a tape mark to delimit the valid data, but the blank tape will stop a reader. *-slow* disables writebehind to make sure the tape is as up-to-date as possible.

```
$some_command | tp -erase -p -slow -conserve
```

This command line does nothing!

```
$tp -noc
```

This example positions the tape to the second file and leaves it there.

```
$tp -nc tskip=2
```

This example copies the next file on the tape to *stdout*. This command leaves the tape at the end of the file you just copied.

```
$tp -noc -g
```

tp

This example copies the previous file to **stdout**, leaving the tape in the position it held before the command.

```
$tp -noc -g tskip=-1
```

This example copies the last file on the tape to **stdout**.

```
$tp -g -append tskip=-1
```

This example rewinds/unloads the tape.

```
$tp -unload
```

DIAGNOSTICS

Can not open tape device '<name>' mode <modes>.

Tape is not write-enabled.

-p and *-erase* require that the tape write-enable be inserted in the tape reel.

Invalid density #.

Can't alter tape density.

The density specified is wrong or not supported by the particular tape controller.

Can not set size of tape block.

The tape controller does not support the block size specified by the *setbs* option.

bs (#) must be multiple of setbs (#).

Unable to allocate # bytes. The system does not have enough memory to supply all the buffer space requested. Try reducing *bs* and *abuf*.

Can not lock program in memory.

You must be the administrator to use **tp** with the Tape Master controller.

Can not open disk file '<name>' in mode <modes>.

Can't open one of the files named on the **tp** command line.

```
tape ->'%s':
```

```
'%s' -> tape:
```

```
'%s' == tape:
```

Normal messages indicating what operation is being performed.

tape retries (apparently recovered).

Reports how often the controller reread or rewrote a block before getting it right. A large number indicates an untrustworthy tape.

TRANSFER COUNT MISMATCH.

More data was read than was written, perhaps because of an error while writing.

Verify error.

bytes different.

Verify count error: disk # tape #.

Disk file longer than tape file.

The file on the tape is different from the file on disk.

system buffers.

bytes, # blocks read from disk. # aver time (us).

bytes, # blocks written to tape. # aver time (us).

bytes, # blocks read from tape. # aver time (us).

bytes, # blocks written to disk. # aver time (us).

Disk and tape files are the same.

Status messages printed only when *-verbose* is specified. Note that there can be more data written to tape than read from disk because of padding inserted to make standard size tape records.

Out of tape.

A write ran over the physical end-of-tape marker, indicating that the tape is too short to hold that much data.

Strange state #, file '<name>'. "Absolutely can not happen."

Tape spfun # failed.

Tape I/O error #.

sio(##,##) = #

#=sio rc

purge_rest: #=rc

purge_rest: Q not empty: #=rc

purge: #=rc

tp

An I/O error, or an attempt to do something not supported by the particular device driver.

NOTE: When the Tape Master controller gives an error, **tp** describes the controller status in great detail. Many of these messages are not errors. For example, **Write ring not present.** is an error only if a write operation is intended.

ALSO SEE

MTS-1 and DK400 Subsystem Users Manual

NOTES

If you use *-put* and *-verify* with **stdin**, the verify will fail because it can not reread the input stream.

All supported tape drives have these properties:

- You can overwrite old data anytime without having to erase the tape. However, any write erases data for an unknown distance beyond the last block written. It may not even be possible to move the tape past this end-of-write point. Therefore, wherever you write on the tape, you should consider that the new end of the tape.
- When **tp** finishes writing to the tape, it writes two tape marks.
- The total capacity of the tape is not accurately known, because tape speed and record spacing varies. If you want to be sure everything will fit, you will have to use much less than the nominal capacity of each tape.
- A physical end-of-tape indicator keeps **tp** from running off the end.

All supported 1/2" tape drives have these properties:

- A standard tape reel holds 2400 feet of tape, that stores about 45 Mbytes of data at 1600 bpi. However, there is a .6 inch gap between records, which reduces the capacity

drastically for short records. For example, with 512-byte records, the tape can store only about 15 Mbytes. To get to 90% tape utilization takes at least 8500 bytes per record at 1600 bpi, and proportionately more at higher densities.

- Changing tape density in the middle of a tape is not supported, so any particular tape has to be written in one density throughout.

The 1/2" tape with the Tape Master controller has the following special properties:

- Only the system administrator can use `tp`, because `tp` has to lock itself in memory to use this controller.
- The `setbs` option can not be used to map a large disk into several small tape blocks. `tp` will always write one tape record (`bs` bytes except the last block) for each buffer-load of data.
- The `-unload` option terminates with an I/O error, because driver does not check why the tape went offline.
- 65535 (64k-1) bytes is the largest tape block size allowed.

The 1/2" tape with the SASI bus controller has the following special properties:

- The `density=#` and `-slow` options act by altering the default name by which `tp` opens the tape drive. Therefore they do not work if the `tape=name` option is
- The `-conserve` option does nothing as this controller never has to stretch the tape gap to maintain streaming.
- The controller has a buffer which can contain up to 64 Kbytes of data that has ostensibly been written but isn't yet on the tape. If the buffer is not forced out (by writing a tape mark or using the `-slow` option) before the tape is removed or the system crashes, the data will be lost.
- 65536 (64k) bytes is the largest tape block size allowed.

tp

- Sizes over 32768 (32k) may prevent the tape from streaming.

The 1/4" cartridge tape has the following special properties:

- A standard cartridge contains 300 feet of tape and holds about 45 Mbytes of data.
- The controller only supports 512-byte records, so in effect the *setbs=512* option is always present.
- *density=#*, *-slow*, and *-conserve* do not work.
- The drive can only stream for *bs* bytes, then it must stop. Therefore, *bs* should be as large as possible.
- The tape can not be moved backwards other than a complete rewind. This prevents using *-append*, *tskip=-#*, or any two of *-p*, *-g*, and *-v* with *-norewind* or *tskip=#*. Also, the tape position after a *-p* is one file mark farther out than described above.
- *-erase* always erases the entire tape.

USES

nothing

translit substitutes characters

FORMAT

translit *from_set* *to_set* [*file1*...*filen*]

DESCRIPTION

translit performs one-to-one character transliteration, changing the characters specified in the *from_set* to the corresponding characters in the *to_set*. Since the mapping is one-to-one, the *from_set* must be equal to or larger than the *to_set*. If the *from_set* is larger, the end of the *from_set* is mapped to the last character of the *to_set* (see below).

The files themselves are not modified; all output appears on standard out. If you do not specify files, **translit** reads from standard input.

translit uses the following notations:

- \b** Represents BACK SPACE.
- \ddd** Represents ASCII code in octal.
- \n** Represents newline (ASCII LF).
- \r** Represents carriage return <cr>.
- \t** Represents TAB.
- ** Represents backslash.

OPTIONS

none

EXAMPLES

Character ranges can be expressed inside square brackets. If the character **^** precedes the square brackets, it means all the characters not in the brackets. For instance, to change every **a**, **b** and **c** in the file *foo* to **A** and write the modified file to standard output, type

```
$translit [a-c] A foo
```

translit

In the next example, all occurrences of **a** will become **w**, **b** will become **x**, **c** will become **y**, **d** will become **y**, and **e** will become **y** since the *from_set* is larger than the *to_set*.

```
$translit [abcde] [wxy] file]
```

To make a list of all the words used in the files *.doc, type

```
$translit "[a-zA-Z]" "\n" *.doc|quicksort|unique >  
wordlist
```

DIAGNOSTICS

none

ALSO SEE

change substitutes text strings in files

NOTES

When the *to_set* is smaller than the *from_set*, multiple occurrences of the folded characters are not generated. Translating all letters, white space and punctuation, to X's, for example, will cause only a single X to appear on the output stream.

If the *to_set* is empty, the characters of the *from_set* generate no output characters; they are filtered out.

You can use reverse ranges; for example, `$translit [a-z] [z-a]` will change every a to z and every z to a and so forth.

USES

nothing

true

true returns a success indication

FORMAT

true

DESCRIPTION

true does nothing except return an exit status of 0. It is primarily useful in conditional expressions in shell scripts.

OPTIONS

none

EXAMPLES

See the UNIX **sh** command.

DIAGNOSTICS

none

ALSO SEE

false returns an error indication

sh UNIX Bourne shell

csh UCB C shell

NOTES

none

USES

nothing

ttymodes

ttymodes sets or displays terminal modes

FORMAT

ttymodes [options][modes]

DESCRIPTION

ttymodes displays or sets the modes on the device associated with standard input, which in most cases is your terminal.

The modes deal with input, echoing and output. Characters are further divided into two major classes: alpha or printable ASCII and control. The alpha characters are those in the range of 'space' (octal 040) through '~' (octal 0176), and octal 0240 through 0376. The control characters are defined as octal values 000-037, 177, 200-237, and 377.

To display the modes currently set on the terminal, type **ttymodes** without any arguments. To alter the modes, type the mode either with or without the preceding up arrow (^). The up arrow before a mode means that the mode is turned off.

OPTIONS

An asterisk before an option means that the option is not supported by the TP-104 or the TP-204.

breaklist=<list of chars> Specifies a list of characters to be recognized as wakeup characters in addition to any wakeup characters implied by the **wakealpha**, **wakectl**, or **wakenl** modes. The default is an empty list. See **NOTES** for a description of ways to enter characters.

-default This switch sets all modes to the standard initial state. This state is shown in the **EXAMPLES** section. Note that the *erase*, *linekill*, *retype*, *stop*, *start*, *kill*, *suspend*, *eof*, *escape*, *breaklist*, *checkparity*, *parity*, *speed*, *wordlen*, *rts*, and *stopbits* options are not affected.

ttymodes

erase=x The erase character (initially DEL) is changed to the character specified. See **NOTES** for a description of ways to enter characters.

eof=<char> Sets the character to be recognized as the end of file character. The default is ^D. See **NOTES** for a description of ways to enter characters.

escape=<char> Sets the character to be recognized as the escape character. The default is ESC. See **NOTES** for a description of ways to enter characters.

ispeed=x* Sets the input baud rate. Our current hardware does not have split speed capability, *ospeed* is used for speed control although *speed=* also works. Note that the *speed* change will only be indicated by **ttymodes, but there will be no actual effect.

kill=<char> Sets the character to be recognized as the process kill character. The default is ^C. See **NOTES** for a description of ways to enter characters.

linekill=x The line kill character (initially ^U) is changed by this option. See **NOTES** for a description of ways to enter characters.

-nowait Forces the system to change the modes immediately. Normally, the system waits until all previous output has been sent to the terminal before changing modes.

**ospeed=x* Sets the output rate and currently the input rate. Although this has an affect, it won't be indicated by *ispeed*; *speed* must be changed in order for both the baud rate and the indication to really change.

-reset Restores modes to default values like the *-default* option. In addition, resets all special characters to their default values, sets the breaklist to an empty list, and resets all hardware control options except *speed* to their default values.

ttymodes

retype=<char> Sets the character to be recognized as the retype input line character. The default is ^R. See **NOTES** for a description of ways to enter characters.

**rts=x* Request to Send. Sets the RS232 signal. Options are on or off.

**speed=x* Sets input and output baud rates to value.

Acceptable values for 'speed=' are:

hangup	50	75	110	134.5
150	200	300	600	1200
1800	2400	4800	9600	19200
38400	external	external16	auto	

Note that external, external16, and auto are not supported. The **ttymode** display will change but the actual baud rate will not.

Note that setting *speed=hangup* sets the RS232 signal "DTR" low; any other *speed* sets it high.

start=<char> Sets the character to be recognized as the restart output character. The default is ^Q. See **NOTES** for a description of ways to enter characters.

stop=<char> Sets the character to be recognized as the stop output character. The default is ^S. See **NOTES** for a description of ways to enter characters.

**stopbits=x* Specifies the number of *stopbits* (the number of bits after the last bit of a character and before the start bit of the next character). Acceptable values for *stopbits=* are: 1, 1.5, and 2.

suspend=<char> Sets the character to be recognized as the process suspend character. The default is ^Y. See **NOTES** for a description of ways to enter characters.

tty= tty_device_file_name allows you to specify a tty file without using IO redirection. Open this file with `O_NDELAY`, so the open will not block (useful for using **ttymodes** on modem-controlled lines, when the usual

ttymodes

ttymodes < /dev/ttyn would hang in the open waiting for carrier).

**wordlen=x* Specifies the number of bits per character. The options are 5, 6, 7, and 8.

Modes

Parity

**checkparity* Check parity, can be either on or off. When on, parity is checked according to the type specified by the *parity* mode. When off, parity is not checked, however, it still may be generated if the *parity* mode is set to other than *none*. This mode follows the *parity* mode — if parity is set to *none*, *checkparity* will be set off; if parity is set to other than *none*, *checkparity* is set on. *checkparity* cannot be set with *parity* set to *none* — if attempted, *parity* will be forced to *odd*.

**parity* Specify one of five options: *even* (even parity), *odd* (odd parity), *none* (no parity), *mark* (force the parity bit to be one), and *space* (force the parity bit to be zero). The only modes supported are even, odd, and none.

Note that changing this option to *even*, *odd*, *mark*, or *space* turns on *checkparity*; setting it to *none* turns it off.

Input Modes *i8bit* This mode allows all bits including the eighth bit (parity) to be passed on input. When set off, the 8th bit is always set to a 0.

icrlf When on, the ASCII carriage return character <cr> is mapped to the UNOS newline character (ASCII line feed, octal 012) on input. When off <cr> is passed as <cr> (octal 015).

irawedit When on, permits the input-editing characters (*erase*, *linekill*, and *retype*) to be read as normal characters; otherwise these characters have the effect of deleting previously typed characters (the erase and line-kill functions) or of redisplaying an input line (retype character).

eof When on, the EOF character is mapped to mean end-of-file. It acts like a wakeup character but does not appear in the data. If there are any preceding characters (not wakeup characters) they become immediately available to the user's program. If there are no preceding characters, the program will receive a zero-length input which is considered an end-of-file.

escape When on, enables escape processing as follows: after typing the escape character, a dollar sign followed by a backspace is echoed placing the cursor on top of the dollar sign. The next character typed bypasses normal special character recognition and is echoed according to the echoing rules established with the *ectl*, *erawctl* and *ealpha* modes.

The mode allows all control characters to be entered literally.

prctl Enables recognition of process control characters (the kill and suspend characters). When off, the kill and suspend characters are not treated as special characters.

scrctl Enables recognition of screen control characters (the stop and start characters). When off, the start and stop characters are not treated as special.

wakealpha Characters are normally kept in an intermediate buffer, which is unavailable to the user's programs, until a newline is typed. Thus a line can be altered with the erase character or linekill before the data is committed with a new line, which is the normal wakeup character.

When on, *wakealpha* causes every printable ASCII character ('space' through '~', octal 040 through 0176, and octal 0240 through 0376) to be a wakeup character. When it is typed, the line so far is made immediately available to the user.

wakeblock When this mode is on it causes a read request to return when a wakeup character is seen or when the count is satisfied; e.g., a read of length one will return as soon as a one character is available whether or not a wakeup character was seen.

ttymodes

wakectl This mode is similar to *wakealpha* but deals with all characters outside the printing ASCII set.

When on, *wakectl* causes any control character to be a wakeup character. Alpha characters are saved in the buffer until any control character is typed. Thus printing characters can still be deleted before they have been committed.

wakenl When this mode is on it causes the UNOS newline character (ASCII linefeed) to be a wakeup character. The default is on.

Echoing Modes

ealpha This controls the echoing of alpha characters. (See *wakealpha* mode.) When on, alpha characters are echoed. When off, no echoing of the alpha characters takes place.

ecrlf When on, causes echoing of a newline character as a carriage return/line feed combination.

ectl When on, this mode enables the control characters to be echoed as [^](LETTER). When off, the *erawctl* mode is examined for echoing of control characters. When on, this mode supersedes the *erawctl* mode.

enobellerr When off, causes a BEL character (audible beep on most terminals) to be echoed in two cases: when the erase character is typed but there is no previous character in the buffer to erase (at the left margin) and when so many characters have been typed that the input buffer overflows.

erawctl When on, this mode enables the control characters to be echoed literally, as typed. If both *ectl* and *erawctl* are off, no echoing of control characters takes place except for those that have special meaning.

etab When on, this mode enables the echoing of the TAB character. How it is echoed is determined by the *oxtab* mode.

ttymodes

printing When on, affects echoing in order to make hard-copy terminals easier to use. The response to the erase character is changed so instead of backspacing over and erasing the deleted character on the screen, it echoes the deleted character with a backslash (\) before and after any sequence of deleted characters. Echoing of the ESC character is changed to eliminate the backspace that is normally put out.

Output Modes

o8bit When on, all characters are passed out normally. If off, the 8th bit is set to 0, prior to output.

ocrlf When on, this mode maps the UNOS newline character (ASCII linefeed) to carriage return/line feed on output. When off, the UNOS newline character is output without a <cr>.

octl When on, this mode forces control characters to be printed as ^(LETTER) on output, except for TAB and newline. When off, control characters are output as sent, except for TAB and newline.

oxtab When on, this mode causes TAB, on output, to be expanded to an appropriate number of spaces, depending on column position, assuming stops every 8 columns. When off, TAB is output literally.

EXAMPLES

These are the default mode settings:

```
$ttymodes
erase=^? linekill=^U retype=^R
stop=^S start=^Q kill=^C suspend=^Y eof=^D escape=^[
^wakealpha ^wakectl wakenl ^wakeblock ^i8bit prctl \
scrctl escape eof icrlf
ealpha ectl ^erawctl etab ecrlf ^o8bit ^octl ocrLf oxtab
^irawedit ^enobellerr ^printing
ispeed=9600 ospeed=9600 wordlen=8 stopbits=1
^checkparity parity=none rts=on
```

To set the modes to the defaults and turn off all echoing, type
`$ttymodes ^ealpha ^ectl ^erawctl ^etab ^ecrlf -default`

ttymodes

To turn on the default echoing, type
`$ttymodes ealpa ectl \erawctl etab ecrlf`

To change the modes on tty2, for example, type
`$ttymodes -default < /dev/tty2`

DIAGNOSTICS

none

ALSO SEE

Introduction: Control Characters

User's Guide to UNOS

UNOS Installation and Management Guide

NOTES

ttymodes performs its changes on the device associated with standard input. It can, however, be redirected to affect other terminals (see examples).

All parameters recognized by **ttymodes** are not necessarily supported by all serial devices on all machines. We have indicated which options are not yet in effect. If you attempt to use an unsupported option you will not be told of its failure.

Characters for the **breaklist**, **erase**, **linekill**, **suspend**, **stop**, **start**, **retype**, **kill**, **escape**, and **eof** options can be entered in any of the following ways:

`^character`

A `^^` followed by any character, `"\0"` will be entered as the ASCII NUL character.

`\n` or `\N`

Will be entered as the newline (ASCII LF) character.

`\r` or `\R`

Will be entered as the return (ASCII CR) character.

ttymodes

`\t` or `\T`

Will be entered as the tab (ASCII HT) character.

`\b` or `\B`

Will be entered as the backspace (ASCII BS) character.

`\f` or `\F`

Will be entered as the form feed (ASCII FF) character.

`\v` or `\V`

Will be entered as the vertical tab (ASCII VT) character.

`\\`

Will be entered as a single `'\'` character.

`\^`

Will be entered as a `'^'` character.

`\<any other char>`

A `'\'` followed by any character not in the above list will be converted into that character, removing the `'\'`.

Any character not preceded by a `'^'` or a `'\'` will be entered unchanged.

USES

nothing

uget retrieves files from a Version 6 UNIX-style file system

FORMAT

uget [-text][-binary][-swab] device [file1...file*n*]

DESCRIPTION

uget reads UNIX version 6 file systems. The filename arguments to the **uget** command are pathnames of files in the UNIX Version 7 file system on the device. The file entries are copied to files with the same name in the user's current working directory. Note that you do not need to mount the device with the **mount** command. Nothing on the disk is updated, only files are extracted.

OPTIONS

-text Reads text files.

-binary Reads binary files.

-swab Swap bytes. This is sometimes needed on floppies generated on DEC equipment.

EXAMPLES

To read the text file, **ferc.c** from **/dev/rdx1**, type

\$uget -text /dev/rdx1 ferc.c

DIAGNOSTICS

none

ALSO SEE

nothing

NOTES

Because of the various interleave schemes used on floppy disk systems, you might first have to "de-interleave" the floppy disk. In addition, since track 0 is sometimes used to hold data even

uget

though it is reserved according to IBM standards, you might have to shift all the data before you can read it with the **uget** command. See the **debe** command (the **iskip** and **oskip** options) for details on doing this.

USES

nothing

umount

umount unmounts a file system

FORMAT

/etc/umount device_file

DESCRIPTION

umount notifies the system to dismount the removable file system mounted via *device__file*.

OPTIONS

none

EXAMPLES

The following sequence unmounts a filesystem from device *nuclear*.

```
shell>umount /dev/nuclear
```

DIAGNOSTICS

Self-explanatory

ALSO SEE

mount mounts a file system

NOTES

none

USES

/etc/mnttab

uname

uname print name of current UNIX system

FORMAT

uname [-amnrsv]

DESCRIPTION

uname prints the system name or, depending on the option chosen, other system information on standard output.

OPTIONS

- a print the information given by all the *uname* options given below.
- m print the machine hardware name.
- n print the node name.
- r print the operating system release.
- s print the system name (default).
- v print the operating system version.

EXAMPLES

```
shell>uname -a  
X X UNOS R7 PRE-REL CP20
```

DIAGNOSTICS

none

ALSO SEE

uname() gets name of current UNOS system

NOTES

none

USES

nothing

unboot is the UNOS bootstrap.

FORMAT

--- **unboot** is a standalone program invoked from the UNBUG ROM ---

DESCRIPTION

unboot is the UNOS bootstrap. It boots all three types of disk controllers: CC-1/VCC-1, Xylogics, and Interphase.

unboot loads a program from the filesystem on any partition on the device where it is installed and starts it executing. **unboot** boots UNOS images from the system disks. For this purpose, **unboot** can load an IOP image into IOP ram, if the processor is so equipped. **unboot** can also save a copy of the first block of a file in a buffer for later use by the operating system (the config file).

unboot understands the structure of UNOS file systems, including UNOS bad block remapping, partitions, partition maps, and hierarchical directories. It knows the format of UNOS executable image files, and can read, use and skip over object file headers in COFF and the old *rel* format. **unboot** uses a small cache to improve throughput on slow devices such as floppies.

You must install **unboot** on the first blocks of a device by the **/sys/copy_boot** command. Arguments passed in the processor registers from the UNBUG ROM code tell **unboot** what files to load, what type processor it is running on, and what controller and device to use.

unboot uses disk drivers built into the UNBUG Rev. 3 ROM and later UNBUG ROMs to actually read blocks from the disk. It contains Xylogics and CC-1 drivers to read the disk when used with UNBUG Rev. 2.7 or older ROMs. All newer devices are supported by driver code in UNBUG rather than driver code in **unboot**.

unboot

For UNBUG Rev. 2.7 and older UNBUG ROMs, **unboot** presumes that UNBUG has already set up the hardware configuration information in the SCSI disk controller it is being booted from. It passes most of the arguments it gets from UNBUG on to the program it loads. (See the UNBUG chapter in the *Universe Installation Manual* for more detail on this).

The sys image (the one specified by the sys argument to UNBUG) is loaded starting at location 0. The text, data, and bss space must not exceed 0x7f000 bytes. If present, the data section must immediately follow the text. The bss, if present, must immediately follow after the data. To do this, use the *-N* option of **ld**. **unboot** initializes the bss space to zero. The IOP image is loaded into IOP ram at an address determined by the processor type, if the processor has an IOP. The sys image file must always be present or **unboot** will abort, the IOP image file must be present only for those processors that have IOP's. The config file need not be present for any processor type. Control passes to the loaded program by simulating a reset, the SP (Stack Pointer) gets the contents of location 0 and the PC (Program Counter) gets location 4.

OPTIONS

none

EXAMPLES

To boot an operating system located on backup root of device cw0 in **/sys/unos.new** and an IOP from **/ioproot** with a config file **/etc/configs/test** type the following in response to the UNBUG boot prompt:

```
cw0,/etc/configs/test,/sys/unos.new,/ioproot
```

To boot a standalone program from file **/mydir/standalone** on partition 2 type

```
cw2,/mydir/standalone,iop.dummy
```

iop.dummy may be any small object file including the iop program supplied with your UNOS (assuming that you do not intend to actually use the IOP to execute it). *iop.dummy* is not needed on processors that do not have IOP's.

DIAGNOSTICS

unboot prints out brief error messages when it encounters an error and then returns to UNBUG. These errors are

<filename> is too big

The IOP file specified by *filename* is larger than the IOP memory (and overwrites itself) or the UNOS file specified is bigger than 71000 (the sum of text, data, and bss sections).

<filename> has bad header

filename is not in *rel* format or COFF format, or the text section doesn't start at 0 with data and bss sections immediately following. For COFF the file must be a 68000 executable file and must start with a text section followed by an optional data section and an optional bss section, in that order.

<filename> not found

unboot cannot find the file specified. Name specified was a non-existent file or directory.

?No FS hdr

The file system header on the device being booted is missing or garbled. Files on device cannot be read.

?bad blocks

This error occurs with Rev. 3 PVH's if

- the bad block record is corrupt
- the bad block map is the wrong version
- the bad block map is too large (maximum of 2000)

For Rev. 2 PVH's an invalid bad block record is treated as indicating that no bad blocks exist.

UNBUG passes a flag to unboot indicating whether UNBOOT should pay attention to bad blocks on the device. This flag is set by Rev. 4 UNBUG to ignore bad blocks on tape and floppy devices; so a bad block list taken from an image copy of a whole disk does not remap records on the tape or floppy device.

unboot

?bad disk addr

The disk address contained in an inode or indirect block is outside the current block partition. This means the file system is corrupt.

?bad partition

The specified partition doesn't exist. This error message also occurs if you specify a partition other than 0 when no partition record or an invalid record exist.

? unknown device

This error indicates an inconsistency between **unboot** and UNBUG. **unboot** tried to open a device name passed to it by UNBUG and failed.

?FTYPE

The file type from the inode entry is not either SMALL or LARGE. This indicates that either a file of some other type was specified or that the filesystem is corrupt

?ioerr

An I/O error was signalled by the controller indicating it could not read some block of the disk.

ALSO SEE

copy_boot copies a bootstrap program to a device

reboot reboots the operating system

UNBUG Chapter, Universe Installation Manual

UNOS Installation Notes

NOTES

none

USES

This program is essential to starting up UNOS.

unique displays repeated or unique lines in files

FORMAT

unique [options] [file1...fileN]

DESCRIPTION

unique reads the named files or standard input and compares adjacent lines and outputs any unique lines and one copy of any redundant lines. For instance, if you have a file that contained a list of words, you could use **unique** to produce a list of only those words that are repeated. Output is sent to standard out.

unique expects a sorted input.

When comparing parts of lines (fields or characters), the entire last duplicate line is the output. When several input files are specified on the command line, the comparison is only of lines within a file; there is no comparison across files.

OPTIONS

c=range Compares character positions, specified by the range.

These are positions within any specified fields (**f=range** option) or in the line as a whole. (If **f=range** is not used, the default field is 1.) The range can be one number (**c=3**), a series of numbers separated by commas (**c=1,3,5**), or two numbers separated by a dash (**c=1-4**).

d=C Specifies input delimiter, where **C** is a single character.

TAB is the default delimiter.

f=range Compares fields, specified by the range. This can be further refined by use of the **c=range** option. (The possibilities for the range are the same as explained under the **c=range** option.)

A field is defined as a string of characters and spaces separated by tabs (or another specified delimiter). Fields are numbered, starting with 1.

unique

- n Prefixes to the line the number of occurrences of the displayed line.
- r Shows only redundant lines.
- u Shows only unique lines. (The default is -r and -u.)
- v Verbose. Prefixes the filename where line was found, along with the regular information.

EXAMPLES

These examples show the difference between the *-r* and *-u* option, used separately and together (the default). The text, as follows, is found in the file *fee*:

```
gear
gear
gear
hard
hare
hear
hear
```

The following command line executes **unique** without options, using the defaults.

```
$unique fee
gear
hard
hare
hear
```

The following command line finds only the redundant lines.

```
$unique -r fee
gear
hear
```

The following command line finds only the unique lines.

```
$unique -u fee
hard
hare
```

unique

The next set of examples use the following text, found in the file *foo*:

```
abc:abs:abx
xyz:abt:abw
abc:zzy:abw
```

The following command line says that fields are separated by colons (*d=:*), and compares only the second character (*c=2*) in the third field (*f=3*). All three lines in *foo* match; only the last line of the selected lines is displayed.

```
$unique d=: f=3 c=2 foo
```

```
abc:zzy:abw
```

The following command line finds that the first two lines are redundant (they both have the same first two characters in the second field) and the third line is unique. The second and third lines are displayed as output.

```
$unique d=: f=2 c=1-2 foo xyz:abt:abw
```

```
abc:zzy:abw
```

DIAGNOSTICS

none

ALSO SEE

common finds identical lines in two different files

difference indicates how two text files differ

quicksort sorts the contents of files

NOTES

Fields and characters are numbered from left to right, beginning with 1.

USES

nothing

unmount unmounts a file system

FORMAT

/bin/unmount device_file

DESCRIPTION

unmount breaks the logical association of a physical file system (device) from the single UNOS logical file system. The files on the physical device are no longer accessible (until remounted).

Within the command, a **sync** is executed to assure that the device is in a consistent state, after which no further access is made. **unmount** fails if the device is in use (which usually means that some user has a current directory on the mounted device or has an open file on the device.) The medium should only be removed after an **unmount** has been done (i.e. floppy).

OPTIONS

none

EXAMPLES

The following sequence unmounts a filesystem from device *nuclear*.

```
$unmount /mnt
```

DIAGNOSTICS

Self-explanatory

ALSO SEE

mount mounts a file system

NOTES

none

unmount

USES

/etc/mounttab To update list of currently mounted devices.

update

update periodically writes out disk cache

FORMAT

`/sys/update [delay=#seconds]`

DESCRIPTION

update runs at regular intervals and causes information encached by the file system to be written out to the disk (if it has been modified). Therefore, if the system should happen to crash, only work done in the last few seconds would be lost. (Some programs, such as editors, don't modify disk files until you exit or explicitly write, so update can't protect you entirely.)

OPTIONS

delay=#seconds Sets the interval between the disk update. This time is 30 seconds, by default.

EXAMPLES

The following example shows the beginning of `/etc/startup`:
echo entering startup file
`/sys/update &`

DIAGNOSTICS

none

ALSO SEE

sync updates disk with in-memory information

NOTES

update is normally started as a background process by the system startup file, `/etc/startup`, when the system enters multi-user mode. If you intend to do much work in single-user mode, you may want to run **update** to prevent losing all your work.

uptime

uptime shows how long system has been running

FORMAT

uptime

DESCRIPTION

uptime displays the amount of time since the system was booted, in days, hours, minutes, seconds, and tenths of seconds.

OPTIONS

none

EXAMPLES

\$uptime

System has been up for 1 day, 0 hours, 29 minutes and 13.4 seconds.

DIAGNOSTICS

none

ALSO SEE

nothing

NOTES

none

USES

/dev/clock

update

USES

nothing

v7uget

v7uget retrieves files from a Version 7 UNIX-style file system

FORMAT

```
v7uget [-binary][-swab][-swabdir][-text] device  
[file1...filen]
```

DESCRIPTION

v7uget reads UNIX version 7 file systems. The filename arguments to the **v7uget** command are pathnames of files in the file system on the device. The file entries are copied to files with the same name in the user's current working directory. Note that you cannot mount the device with the **mount** command. Nothing on the disk is updated, files only are extracted.

OPTIONS

-binary Does nothing.

-swab Swaps all bytes.

-swabdir Swaps bytes only on directories.

-text Swaps all bytes (same as *-swab*).

EXAMPLES

```
$v7uget -text /dev/rdx1 ferc.c
```

Reads the text file, *ferc.c*, from **/dev/rdx1**.

DIAGNOSTICS

none

ALSO SEE

nothing

v7uget

NOTES

Because of the various interleave schemes used on floppy disk systems, you might first have to "de-interleave" the floppy disk. In addition, since track 0 is sometimes used to hold data even though it is reserved according to IBM standards, you might have to shift all the data before you can read it with the **v7uget** command. To do this see the **debe** command (iskip and oskip options).

USES

nothing

version

version displays creation date and time of object files

FORMAT

version file1...filen

DESCRIPTION

version displays the date and time the object modules were created. Use this command to determine the date of a release of a program.

OPTIONS

none

EXAMPLES

```
$version /bin/basic  
Fri Nov 6 1981 15:08:06 /bin/basic
```

DIAGNOSTICS

version: File does not exist. Can not open 'file name'.

version: Bad header in 'file name'.

The file is not a UNOS object file.

ALSO SEE

nothing

NOTES

The assembler and linker put the creation date into the object code header. Modules created before Nov. 3, 1981 do not have this information and therefore are listed as created on Nov 3 1981 with the warning ****OLD***** in the output.

version

USES

nothing

wait

wait waits for process to exit

FORMAT

wait [*pid1* ... *pidn*]

DESCRIPTION

wait causes the Command Interpreter to wait for child processes *pid1* ... *pidn* to exit before accepting another command. If no *pid* is specified, the Command Interpreter waits for the first child that exits.

OPTIONS

none

EXAMPLES

\$wait 115

Waits for process 115 to exit.

\$wait

Waits for any child process to exit.

DIAGNOSTICS

no children There were no active **child** processes.

ALSO SEE

command executes a command line or file

resume resumes execution of a suspended process

NOTES

An attempt to wait for a process that is not a descendant of the Command Interpreter executing the **wait**, causes the Command Interpreter to wait for any and all children (if any) and prints the message, **no children**. This is a built in command to **command**.

wait

USES

nothing

wall

wall sends a message to all users currently logged on

FORMAT

wall [message]

DESCRIPTION

wall writes the **message** to all logged in users who have enabled message writing (that is, have not set the **message** command to *off*. It is the same as the **write** command except that it sends the **message** to *all* users, not just the ones specified.

OPTIONS

none

EXAMPLES

```
$wall "Who wants to go to lunch?"
```

Writes this message on all terminals:

```
From dianne@ 15:08:27 Who wants to go to lunch?
```

DIAGNOSTICS

none

ALSO SEE

who displays names of users currently logged on

write sends a message to specified users

NOTES

wall lists the names of users whose terminals are message-protected.

Remember to quote any meta characters that may be part of the message.

wall

The **wall** command is in the **/etc** directory .

USES

/tmp/login finds out who is on the system.

wcc compiles C programs using the Whitesmiths C compiler

FORMAT

wcc [options] file1..filen

DESCRIPTION

wcc compiles and links a C program using the Whitesmiths C compiler.

As a convention, C programs have a *.c* suffix; you do not need to use the *.c* suffix when you execute **wcc**. If a file name does not end in *.c*, *.m* or *.j*, **wcc** will append *.c* to it and continue.

If you execute

```
$wcc foo
```

foo will contain the executable code.

wcc calls **pp**, **p1**, **p2** and **asm**, which compile the program. **wcc** also calls **link** which links the program.

If you specify a file with a *.m* extension, **wcc** assumes it's an assembly source file and will not call **pp**, **p1** or **p2**. If you specify a file with a *.j* extension, **wcc** assumes it's a object file that only needs to be linked. In this case, **wcc** doesn't call **pp**, **p1**, **p2** or **asm**.

wcc always specifies an output file name to the linker, so the output does not default to *link.out*. If the *o=* option is not used the default is the first file name given to **wcc** stripped of its *.c*, *.m* or *.j* suffix.

wcc creates the *.j* file in the working directory even if the source file is in another directory. This is consistent with the Pascal driver and with what the **build** program expects.

wcc recognizes many of the linker options, including *ssize=*, *-sbt*, *-lun0* and *-uv* and the very primitive *tbase=*, *dbase=* and *sbase=*. **wcc** passes any number of arguments to the linker not just a maximum of 128.

WCC

wcc returns a correct exit code (for the benefit of the **build** program). A 0 returns if it succeeds or an appropriate error code (whatever code is returned by one of the steps it execs).

NOTE: **pp**, **p1**, **p2** only return TRUE/FALSE, so **wcc** returns 1 if one of these returns FALSE.

If any step can not be executed to normal completion (i.e. fork failure, exec failure, step suspends, or is killed), **wcc** stops immediately, prints at least one message and returns a non-zero error code.

wcc always puts */lib/C68rt.j* at the front of your program which sets up the arguments to main and calls main().

wcc calls the Whitesmiths version of the UNOS library (*/lib/unos.wlib*). If you specify any libraries, using the *l=libraries* option, **wcc** scans the UNOS library first.

OPTIONS

-c Compiles only.

Stops after calling **asm**. The output is a *.j* file.

d=defn

D=defn Defines.

Provides a means of passing a define to the preprocessor.

DIR=directory Adds the directory to the search list.

A debugging option that lets you add as many directories as you want to the search list. Each directory name, however, must end in a slash.

-f Uses the floating-point processor.

When you specify this option, the compiler generates code to access the Sky FFP (Fast Floating-point Processor) for arithmetic on float and double types. If any *.j* file is compiled with this option, the link step must also be done to get a different standard library which includes code to initialize the Sky FFP.

- fn* Adds the name of the file being compiled to the beginning of the error message. The name of the file is always followed by a colon (:). This option is the default. To disable it use *-nofn*.

- i=dir* Adds directories to the *#include* search path.
These added directories are searched in the order given, then the standard */include*, to find include file names which are not absolute pathnames. Each directory name must end in a slash.

- l=library* Adds libraries.
The user can specify the pathname of additional libraries other than those **wcc** usually uses.

- lun0* Denotes the default linker option. Prior to Rev. 5, the default was *-no_lun0*. When this option is on, logical unit numbers and UNIX fildes numbers are the same. To override the default, specify *-no_lun0*.

- m* Checks structure membership.
Structure members must belong to the structure to which they are applied. By default this is not checked.

- no_m* Does not check structure membership.
Specified this option to prevent the checking which will be enabled in a future release.

- M* Does not check structure membership.
This option is the opposite of *-m* and is the default. *-m* is included so you can indicate an explicit requirement that structure elements be global in Makefiles.

- no_j* Substitutes *.o* and *.s* for *.j* and *.m* suffixes.
This option may make it easier to convert UNIX Makefiles. In a future release, this will be the default and the *-j* option will substitute *.j* and *.m* for *.o* and *.s*.

- o=file* Names output file.
Use this option to specify a file other than the default output.

WCC

-O Optimizes.

This option does nothing, but is accepted for compatibility with UNIX cc.

-S Produces assembly language instead of object code.

With this option, **wcc** does not run through **asm**, but stops after **p2**. The output has a *.m* extension and is assembly language source.

V=version Adds the version to the suffix list.

A debugging option with no limit to the number of versions you may add to the suffix list.

EXAMPLES

```
$wcc d=DEBUG program.c
```

The compiler compiles lines found between *#ifdef DEBUG* and *#endif*, a common way of including optional debugging statements in a program.

```
$wcc -c file.c  
output to file.j
```

file.j is the unlinked binary object file.

```
$wcc d=name file.c
```

The user can define the preprocessor symbol.

```
$wcc d=name=value file.c
```

The user can define the symbol and use its value.

```
$wcc o=file file.c
```

The file becomes the executable program.

```
$wcc -S file.c
```

Assembly language outputs to file.m

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory.

ALSO SEE

link Binds object modules

gcc Compiles a C program using the Green Hills C compiler.

C Reference Manual by D.M. Ritchie

Programming in C -- A Tutorial by B.W. Kernighan

NOTES

See the C release notes for information about C implementation on UNOS.

wcc automatically defines the following symbols for the preprocessor stage of the compile: M6800, unos, __WHITESMITH.

USES

Nothing

when

when executes a command line at a given time

FORMAT

when [-sh] [-csh] [priority=##] time [date] [file]

DESCRIPTION

when takes command line arguments to determine the time when a command line will run. It then copies standard input to a temporary file or copies a given file name. The file is placed in the spool directory for the **when** command, with a **cd** to the current directory and assignments to all the environment variables prepended. The resulting command file runs under the effective user-id of the person executing **when**, in the current working directory. The device is the standard input, output, and error streams.

Acceptable **when** forms for times are

#

An hour when the command will run (e.g., 3:00 or 15:00). The hour will be interpreted in 24-hour time.

#am

#pm

An hour as above, from 1 to 12, with morning/afternoon disambiguated. 15:00 is 3pm.

##

##am

##pm

A time specified in hours : minutes. May also be followed with a meridian specifier (am/pm). The colon is optional if at least 3 digits are given.

+#

minutes from "now", where "now" is computed as the next integral minute (e.g., I run "when +5" at 15:34:23, and **when** queues for 15:40).

when

+#:##

hours, ## minutes from "now" as above.

With two or more arguments, **when** assumes the arguments after the first are dates to run the command. The date formats are
+# In # days.

week One week from today.

DAY

DAY week On the next following DAY (e.g., Tuesday). Only the first three letters of the day name are checked, without distinction of case. "week" once again indicates one week further into the future.

MONTH #

For example, December 5 (month checked as for days). If the date given is earlier in the year than today, the date is assumed to be next year.

MONTH # #

For example, Jan 1 1984, if you give only two digits for the year. Years less than 80 are prefixed by 20 (i.e., 2010); years greater than 80 are assumed to be 1900's. You cannot specify years before 1980.

MONTH

MONTH

Similar to above, for European format dates. See above for specification of year.

#/#

Month/day, year chosen as above.

#/#/#

Month/day/year, year parsed as above.

Note that in all cases, when a time is ambiguous, **when** uses the current date to choose an unambiguous time in the future to run the command line.

when

Use the priority argument to mark **when** entries which run before or after other arguments for a given time. The range of priorities is 0 - 19, with 0 having the most priority. The default priority is 30. Only the administrator may specify priorities less than 10.

OPTIONS

-co, **-sh**, and **-csh** indicate the command processor used to run the resulting script. These options also alter how the environment variables are output, in order to conform to the syntax required by each. The default is **-sh**; UNOS users will usually want to specify **-co**. Alternatively, the environment variable SHELL may be set to one of *csh*, *sh*, or *command*.

EXAMPLES

The following example sets up a greeting for everyone logged on next New Year Eve.

```
$when 00:00 1 Jan wall "Happy New Year!" <EOF>
```

This next example uses **when** to perform a task recursively; **who** runs three times every hour. It sets up a command file *whofile* that is executable and in the current working directory when **when**. *whofile* runs itself every hour (it specifies +19 minutes to account for the fact that it will take some time to start the command running, and **when** schedules using the next minute).

```
$cat > whofile ; filemodes rwx r whofile who >>  
/usr/adm/who_records echo whofile | when +19 <EOF>
```

```
$when +19 ./whofile <EOF>
```

DIAGNOSTICS

none

ALSO SEE

chron performs periodic commands specified in file
whenq inspects queue of **when** requests
whenrun runs queued **when** requests

when

NOTES

Since **when** only queues a request, this implies that there is a program which runs these requests. That program is **whenrun**, which **chron** invokes periodically. Depending on how often **chron** invokes **whenrun**, requests may be executed later than the precise time specified.

Before using **when** for the first time, the system administrator must make sure that the directories **/usr/spool** and **/usr/spool/when** exist. This is a one time activity and both directories can be created with the **makedir** command.

USES

/lib/when-queue.??? queue of when requests

/usr/spool/when/YYYY.D.MM.HHHH.## spooled script file

whenq

whenq inspects queue of **when** requests

FORMAT

whenq

DESCRIPTION

whenq prints out the current queue of **when** requests. These requests may be cancelled by deleting the associated request files.

OPTIONS

none

EXAMPLES

In this example, the **usr buffer** section appears on a separate line from the rest of the information. If you try this example, the **usr buffer** section appears on the same line as the rest of the information. Due to space limitations, **/sp** in the example represents **/usr/spool/when**.

\$whenq

```
RQID PRI Status Time of Request\  
      user buffer  
  8  30 WAIT   Tue Dec 20 1983 17:00:00 107\  
      </sp/1983.11.20.1700.0>  
  9  30 WAIT   Tue Dec 20 1983 19:10:00 107\  
      </sp/1983.11.20.1910.0>  
 10  30 WAIT   Sun Jan  1 1984 12:00:00 107\  
      </sp/1984.0.1.1200.0>
```

DIAGNOSTICS

none

whenq

ALSO SEE

chron performs periodic commands specified in file
when executes a command line at a given time
whenrun runs queued **when** requests

NOTES

none

USES

/lib/when-queue.??? queue of when requests
/usr/spool/when/YYYY.D.MM.HHHH.## spooled script file

whenrun

whenrun runs queued **when** requests

FORMAT

whenrun

DESCRIPTION

whenrun examines the queue **/lib/when-queue**, performing those requests whose time has come. Perform the requests by running the desired command interpreter on the spooled buffer file. See the **when** command for more detail.

OPTIONS

none

EXAMPLES

DIAGNOSTICS

none

ALSO SEE

chron performs periodic commands specified in file
when executes a command line at a given time
whenq inspects queue of **when** requests

NOTES

As a security measure, **when** requests are queued as unwritable to anyone except the originator of the request. **whenrun** refuses to run a script if the owner of the buffer is not the originator of the request.

Run **whenrun** from **chron** every once in a while. Between 5 and 30 minutes is typical, depending on the desired granularity. See **chron** for an example.

whenrun

Before using **when** for the first time, the system administrator must create the directories **/usr/spool** and **/usr/spool/when**; this is a one time activity. Create both directories with the **makedir** command.

USES

/lib/when-queue.??? queue for when requests

/usr/spool/when/YYYY.D.MM.HHHH.## spooled script file

which

which shows which file will be executed for a name

FORMAT

which namel ... namen

DESCRIPTION

which takes each name in its argument list in turn and duplicates the action of a command interpreter in searching for an executable file. If an executable file is found its pathname is printed. Specifically the procedure is:

- The environment variable "SHELL" is examined to determine which command interpreter you are using.
- If "SHELL=/bin/command" the name is checked to see if it is an abbrev. If it is the name of an abbrev, the abbrev definition is printed in the same format that the '#l' command uses, except that an 'l' or 'g' is appended to the '#b' to indicate whether it is a local or global abbrev.
- If "SHELL=/bin/csh" the name is checked to see if it is an alias defined in .login or .cshrc in your home directory. If it is an alias the definition line from .login or .cshrc is printed.
- If "SHELL" is not "/bin/command" or "/bin/csh" or is not defined or if the appropriate procedure above has not matched the name, the directories defined in the "PATH" environment variable are searched in turn to see if any of them contain an executable file with a matching name. If one is found its pathname is printed and the search stops for that name.
- If none of the above procedures finds a match for the name a message is printed:
which: <name> not found.

which

OPTIONS

none

EXAMPLES

```
$which print test selcop dungeon list noname
#bg print do "prfmt -F $r1 | lpr -n alias=$1" /* abbrev */
./test /* executable file in current directory */
/usr/wjr/bin/selcop /* executable file */
/usr/bin/dungeon /* executable file */
/bin/list /* executable file */
which: noname not found. /* no executable file */
```

DIAGNOSTICS

Error messages will be printed if the "HOME" or "PATH" variables are not in the environment.

ALSO SEE

nothing

NOTES

If you are using the C shell ("SHELL=/bin/csh") only aliases defined in the *login* and *.cshrc* files can be found. Aliases defined from a terminal with the 'alias' command can not be found. as aliases.

If you are using the "command" shell, abbrevs that are "pushed" are not found as abbrevs.

USES

locals in current directory if SHELL=/bin/command.
globals in home directory if SHELL=/bin/command.
login in home directory if SHELL=/bin/csh.
.cshrc in home directory if SHELL=/bin/csh.

who

who displays names of users currently logged on

FORMAT

who [arg1 arg2]

DESCRIPTION

who displays a list of all users currently logged in. For each logged on user, a line is displayed which contains the tty the user is logged on to, the user name, and the time that the user logged on.

Given any two arguments, **who** prints your user name on the standard output (see the example).

OPTIONS

none

EXAMPLES

```
$who
tty0 dianne Tue Sep 18 1984 15:26:06
tty2 ken Tue Sep 18 1984 11:29:27
tty5 ewheeler Tue Sep 18 1984 09:13:33
```

```
$who am i
ken
```

DIAGNOSTICS

none

ALSO SEE

ps displays process status
wall sends a message to all users currently logged on
write sends a message to specified users

who

NOTES

If a user uses the **setuser** command to change who they are, they are still listed as who they logged on as, not who they "setuser" to.

USES

/tmp/logins To find out who is on the system.

write

write sends a message to specified users

FORMAT

write [options][message] user(s)

DESCRIPTION

write sends a message to the specified user's terminal. If the user is not logged on, the message is not sent.

If you specify several users, separate their names with commas. You can use distribution lists instead of login names (see the **email** command).

If you do not specify a message, **write** prompts you for lines of text, as if you had used the *-interactive* option. It sends each line until you type **^D** or press the RETURN key on an empty line. This allows you to send multi-line messages or have conversations.

If you are sending messages across the network (see *UniverseNet Administrator's Guide*), you do not have to specify the system the receipt is on. The **/etc/passwd** file contains users' home systems. If the user you're writing to is logged onto a system other than their own, specify that system by typing **@systemname** as part of the user name; i.e.,
write jeff@orc

By default, **write** prepends the sender and the time to all messages, and informs the sender if there is no recipient.

OPTIONS

All options may be abbreviated to their first letter.

-help Prints to standard error a few lines which list all the flags which the **write** command will accept.

write

- interactive* Prompts the user for more messages. To indicate the end of the message, type ^D or press the RETURN key on an empty line.
- noheader* No header on the message. A message is sent without the sender and time header.
- oneheader* One header on the message. Used when engaged in a conversation (using the *-interactive* option). The first line of the message has a header, all succeeding lines do not.
- silent* Prints no messages to standard out except in case of a system error. For instance, if someone is not logged in, you won't be informed. No exit message is printed.

EXAMPLES

dianne types:

```
$write jeff "What about lunch?"
```

On jeff's screen appears:

```
From dianne 12:45:09 What about lunch?
```

Another user types:

```
$write jeff -noheader "What about lunch?"
```

On jeff's screen appears:

```
What about lunch?
```

dianne types:

```
$write jeff -oneheader >How about lunch? >I think it is  
about time.
```

On jeff's screen appears:

```
From dianne 12:45:09 What about lunch? I think it is  
about time.
```

write

To send a message to everyone on the software and documentation distribution lists, type:

```
$write software, documentation -silent "Lunch?"
```

Because the *-silent* option was used, the sender will not be informed if someone in either group has already logged off and gone to lunch.

DIAGNOSTICS

write: users or dls ____" do not exist

ALSO SEE

email sends and receives electronic mail

message allows or prevents writes to terminal

wall sends a message to all users currently logged on

NOTES

If the recipient is logged on to more than one terminal, **write** sends a message to every terminal on which the recipient is logged in.

If the *-interactive* option is not used, no prompts are displayed on the terminal. This means that you can execute a command line such as:

```
$cat file | write horatio
```

The file will be written to user horatio without prompts in between lines.

If you have UniverseNet, the **write** command can be used over the network. See the *UniverseNet Administrator's Guide* for details.

USES

/tmp/logins To check who is currently on the system.

/etc/maillists/ For the location of the global distribution files.

.maillist.* In the home directory for the local distribution files.

/etc/passwd To validate user names.

write_bad_blocks

write_bad_blocks creates/overwrites bad block list on disk

FORMAT

/sys/write_bad_blocks input_file device_path

DESCRIPTION

write_bad_blocks reads the input file to obtain a list of block addresses which the system should not use, and writes the list onto the disk device.

The input file contains two kinds of lines:

- A line containing two numbers separated by spaces (or tabs) represents the cylinder number and track (head) number of a known defect. The numbers are decimal, unless they start with 0 (octal) or 0x (hexadecimal).
- A line beginning with the double quote character (") is a comment line; **write_bad_blocks** prints the line (on standard output) and otherwise ignores it.

OPTIONS

none

EXAMPLES

\$p bblast

"Bad block list for 40-MByte Quantum disk drive.

28	0
76	1
125	1

\$write_bad_blocks bblast rcqall

"Bad block list for 40-MByte Quantum disk drive.

Bad block record (3 items) written on 40M_Quantum_Winchester /dev/rcqall.

In this example, the defects are in track 0 of cylinder 28, track 1 of cylinder 76, and track 1 of cylinder 125.

The bad block record is written successfully.

write_bad_blocks

```
$write_bad_blocks bblast /dev/rcqall
"Bad block list for 40-MByte Quantum disk drive.
write_bad_blocks: Device /dev/rcqall already has a bad block record.
Do you want to overwrite it with a new one? Answer YES or NO:yes
Bad block record (3 items) written on 40M_Quantum_Winchester /dev/rcqall.
This gives the user a chance to back out of a command
when the target disk does not seem to be a new one--perhaps the wrong
disk is in the drive.
```

DIAGNOSTICS

Self-explanatory error messages display if the input file is not in the right format, or if file system errors are encountered.

ALSO SEE

print_bad_blocks displays bad block list
UNOS System Concepts Manual

NOTES

Use **write_bad_blocks** with raw devices (i.e., unbuffered).
Using a buffered device will fail with the error message:
device xxx is not a disk

The entire track containing each defect is marked unusable, since the algorithm for mapping position within a track to block number is hidden within the disk controller so that we can use many disk controllers with the same mapping software.

Some disk drives supported by UNOS are formatted with 256-byte sectors and an odd number of sectors per track, for example, the 10 MByte Winchester on a Universe 68/05 system. For these disk drives, map out two tracks to make the defect size an integral number of blocks.

The bad block record should not be altered on a disk which contains a file system because then the data would appear to be at different locations, resulting in scrambling most of the files and directories. The only way to recover the file system would be to put the bad block record back the way it was when the

write_bad_blocks

file system was initialized with the **makefs** command. If you have doubts about the coherency of the file system, the file system should be re-initialized (all files deleted) by using the **makefs** command.

UNOS stays within the limit enforced by the disk driver.

USES

none

write_partition

write_partition creates/overwrites the partition record for a hard disk

FORMAT

/sys/write_partition [offset=block_number] partitions
device

DESCRIPTION

write_partition initializes the partition record of the hard disk specified as *device*. The partition record is a record on the hard disk which specifies the size and location of the partitions on the disk. The partition map is specified in the file named by the *partitions* argument.

The *partitions* file format consists of the following:

- A comment line specified with "(double quotes) as the first character of the line.
- A partition specification line containing three numbers. There are two ways to use **write_partitions**:
 1. The first number is from 0 to 30 and specifies the partition number, which corresponds to the low bits of the minor device number in the device node which will access this partition.
 2. The second number specifies the starting location of the partition in blocks.
 3. The third number specifies the limit of the partition in blocks. The limit of the partition is the first block past the partition.
- If the starting location of the partition is equal to the limit of the partition then the partition is of zero length and is inaccessible.

write_partition

The descriptions of the partitions are read from the file specified by **partitions** and the resulting table written to the named device.

The *partitions* file format:

A comment line may be specified by having a " (double quote) as the first character of the line.

A partition specifications line contains three numbers. The first number is from 0 to 30 and specifies the partition number, which corresponds to the low 5 bits of the minor device number in the device node which will access this partition.

The second number specifies the starting location of the partition in blocks. The third number specifies the limit of the partition in blocks. The limit of the partition is the first block past the partition.

If the starting location of the partition is equal to the limit of the partition then the partition is of zero length and hence is not accessible.

OPTIONS

offset=block+number Adjusts the starting offset of partition 0. If *offset* is used with a partition file the partition map is first set up from the file and then the offset is applied to partition 0. If *offset* is used without a partition file, the existing map on the device is adjusted. If no partition file is specified and there is no existing partition map on the disk an error message is returned. This option facilitates the conversion of partition maps to the current UNOS release.

write_partition

EXAMPLES

The following is the normal cqall partitioning:

```
#!/sys/print_partition /dev/rcqall
```

Disk type=cq0_quantum_40M has the following partitions:

"partition	offset	limit
0	0	2000
1	2000	4000
2	4000	24000
3	24000	44000
4	44000	64000
5	4000	44000
6	64000	68000
7	0	69632

#

Partition 6 is a 4000 block area which may be used as a swap area or a virtual floppy. For a single user system it is decided that the 4000 block area which is partition 6 should be made into 2 2000 block partitions. The file *parts* is created by:

```
#!/sys/print_partition /dev/rcqall > parts
```

#

And edited to look like this:

```
#p parts
```

Disk type=cq0_quantum_40M has the following partitions:

"partition	offset	limit
0	0	2000
1	2000	4000
2	4000	24000
3	24000	44000
4	44000	64000
5	4000	44000
6	64000	66000
7	66000	68000

#

Then the new partition map is written to the disk by:

write_partition

```
#/sys/write_partition parts /dev/rcqall
```

```
#
```

DIAGNOSTICS

Messages are self-explanatory. If the disk already has a partition record, **write_partition** asks if you want to overwrite it, and will overwrite it only if you answer Y or YES.

ALSO SEE

print_partition displays the partition record for a hard disk

NOTES

The partition record is stored in the first block of the disk. The first twelve blocks of a hard disk that contains the PVH, partition map, bad block map, and the bootstrap routine, are accessible only through the special partition, partition 31 **/dev/r[xx]all** because partition 0 starts at block 12.

USES

nothing

write_pvh

write_pvh places a physical volume header on the specified disk device.

FORMAT

/sys/write_pvh pvh_file device

DESCRIPTION

write_pvh places a physical volume header on the specified disk device. This allows the disk driver to more easily determine the characteristics of the selected disk, and assures compatibility with future releases of UNOS.

The first argument names a file which contains a readable specification of the disk parameters. The format is described below. A number of files in **/sys/pvh** contain default specifications for standard disks — for example, **/sys/pvh/cq** for the 40 Mbyte Quantum drive used in the UV-68/35.

The second argument is the name of a disk device on which a physical volume header is written. This must be an unbuffered device which begins at the beginning of the physical disk (i.e., **/dev/rcq0**).

The *pvh_file* format includes:

- A comment line specified by having a " (double quote) as the first character of the line. The first comment line is stored in the pvh, and can be displayed with **print_pvh**.
- An attribute specification line contains an attribute name, a colon, and a numeric value. Attribute names are:
 - * *cylinders*, how many cylinders
 - * *heads*, how many tracks per cylinder
 - * *sectors* how many sectors per track
 - * *bytes* how many bytes per sector
 - * *reduce* reduced current cylinder
 - * *stepperiod* time between step pulses
 - * *stepwidth* duration of step pulse
 - * *stepmode* 0 for constant step rate

write_pvh

- * *controller* printable controller type
- * *interleave* size code for DTC-900 controller
- * *recovery* the time, in sectors, between the end of one operation and the start of the next
- * *option1, option2, option3* the meaning of these fields is dependent on the particular device and controller

OPTIONS

none

EXAMPLES

To install a physical volume header on a 32 megabyte Winchester disk, use the following command:

```
#/sys/write_pvh /sys/pvh/cd /dev/rcdall  
#
```

DIAGNOSTICS

Messages are meant to be self-explanatory. If the named device already has a pvh, **write_pvh** asks if you want to overwrite it, and will overwrite it only if you respond Y or YES.

ALSO SEE

print_pvh displays a physical volume header from a disk
UNOS System Concepts Manual

NOTES

Future releases of UNOS will require physical volume headers on all disks.

For the purposes of physical volume headers, floppies are not disks.

The **/sys/pvh** directory contains correct physical volume headers for all disks supported by Charles River Data Systems.

Although you do not have to be the system administrator to write a volume header to a disk, in practice, the disk devices

write_pvh

should not permit access to anyone else.

Always use the raw disk device.

This command is not applicable to floppies and tapes.

USES

nothing

yes

yes writes a word repeatedly to standard output

FORMAT

yes [*arg*]

DESCRIPTION

yes, if *arg* is unspecified, repeatedly writes "yes", followed by a carriage return, to standard output. If *arg* is specified, *arg* repeats rather than *yes*. *yes* repeats until interrupted.

OPTIONS

none

EXAMPLES

In the following example, *erase* erases a file structure one structure at a time. Because many file structures may exist and because *erase* prompts for confirmation, **yes** can provide *erase* with a form of *carte blanche*.

```
$>yes | erase
erase hard disk #1 (y/n): yes
hard disk #1 erased.
erase hard disk #2 (y/n): yes
hard disk #2 erased.
erase hard disk #3 (y/n): yes
hard disk #3 erased.
etc.
```

DIAGNOSTICS

none

ALSO SEE

nothing

yes

NOTES

none

USES

nothing

Commands Functional Grouping

GROUP 1

chgrp
chmod
chown
contig
copy
delete
list
makedev
makedir
move
nice
notify
pmask
setmask
setpri

GROUP 2

advance
backup
confedit
copy__boot
debe
diskusage
download
fmtdisk
format
fscheck
iostats
installation
makefs
makeec
mount
package
printec

reboot
restore
shutdown
spfun
tp
unboot
unmount
uget
v7uget
write
write__bad__blocks
write__partition

GROUP 3

cat
echo
describe
hexodec
lpd
lpmaint
lpr
p
prfmt
qfix
qmaint
qprint
tail
tee
yes

GROUP 4

change
cut
find
match

paste
translit
ted

GROUP 5

chgpw
cpw
id
login
newgrp
setuser
su
uname

GROUP 6

command
csh
false
getopt
kill
ksh
pound__sign
printenv
ps
repeat
resume
sh
sleep
suspend
true
wait

GROUP 7

chron
sync

Functional Grouping

update	GROUP 10	print__pvh
when	email	printlog
whenrun	message	setdate
	wall	space
GROUP 8	who	sysversion
ar	write	time
asm		
build	GROUP 11	uptime
casm	frecv	version
cdeb	ftrans	whenq
ccom68	mnett	
cofftorel	mnetr	
debug	suname	
dump	term	
gcc	ttymodes	
ld		
link	GROUP 12	
lorder	archive	
makedev	calltree	
nm	cd	
reltocoff	genlib	
size	hist	
strip	log	
sysgen	logger	
unlink	profile	
wcc	pwd	
GROUP 9	GROUP 13	
common	conftable	
compare	date	
count	iostat	
difference	ipcstat	
quicksort	lpq	
unique	print__bad__blocks	
	print__partition	

