The OS-9 Utilities

System Command Descriptions

This chapter contains descriptions and examples of each of the OS-9 command programs. While you generally execute these programs from a shell command line, you can also call them from most other OS-9 programs.

At the time of this edition, OS-9 supports 78 utilities and built-in shell commands. They range from commonly used functions such as dir, chd, and copy to advanced system management tools such as dcheck and iniz. For quick reference purposes, the format of the information on the utilities is standardized. Each utility has a section concerning syntax, function, options (if any), examples, and any special uses.

The utilities are broken down into three categories:

- **Basic Utilities:** Every user should become familiar with these utilities. Many of them have been discussed in the earlier chapters because of their importance.

attr	backup	build	chd	chx	copy	date
del	deldir	dir	dsave	echo	edt	format
free	help	kill	list	makdir	merge	mfree
pd	pr	procs	rename	set	setime	shell
w	wait					

¡ **Programmer Utilities:** These utility programs are extremely helpful to the intermediate or advanced programmer. They allow greater exploration of OS-9's timesharing environment and more dynamic file manipulation.

binex	cfp	cmp	code	compress	count	dump
ex	exbin	expand	frestore	fsave	grep	load
logout	make	printenv	profile	qsort	save	setenv
tape	tee	touch	tmode	tr	unsetenv	

¬ **System Management Utilities:** These utility programs are used primarily by system managers and advanced assembly language programmers. Beginning programmers rarely need to use these commands:

break	dcheck	deiniz	devs	diskcache	events	fixmod
ident	iniz	irqs	link	login	mdir	moded
os9gen	romsplit	setpr	sleep	tapegen	tsmon	unlink
xmode						

Formal Syntax Notation

Each command section includes a syntactical description of the command line. These symbolic descriptions use the following notations:

[ ] = Enclosed items are optional

{ } = Enclosed items may be used 0, 1, or many times

< > = Enclosed item is a description of the parameter to use:

<path>	=	A legal pathlist
<devname>	=	A legal device name
<modname>	=	A legal memory module name
<procID>	=	A process number
<opts>	=	One or more options specified in the command description
<arglist>	=	A list of parameters
<text>	=	A character string ended by end-of-line
<num>	=	A decimal number, unless otherwise specified
<file>	=	An existing file
<string>	=	An alpha-numeric string of ASCII characters

General Notes

- The utility syntax specified in the command section does not include the shell's built-in options like alternate memory size, I/O redirection, piping, etc. The shell filters out these options from the command line before processing the program being called.
- The equal sign (=) used in many utility options is generally optional. The k used in the alternate memory size option is also generally optional. For example, you may write -b=256k as -b256, -b256k, or -b=256.
- Utilities that use the -z option expect one file name to be input per line. If you use the -z=<file> option of a utility, <file> may contain comments.
- Unless otherwise specified, command line options may appear anywhere on the command line. For example, the following command lines provide the same results:

  attr -a junk -pw
  attr junk -a -pw
  attr junk -pw -a

- Utilities with only the -? option do not allow you to list any other options on the command line. Also, built-in shell commands, such as chd and set, do not have any options including the -? option.
- cio, the utility trap handler, must be in the execution directory or pre-loaded into memory. By using special I/O techniques, cio allows the utilities to be much smaller. Most utility programs fail to execute if cio is missing. cio is typically loaded into memory at startup.

attr — Change/Examine File Security Attributes

SYNTAX: **attr [<opts>] {<path>} {<permissions>}**

FUNCTION: attr is used to examine or change the security attributes (<permissions>) of the specified file(s).

To use the attr utility, type attr, followed by the pathlist for the file(s) whose security permissions you want to change or examine. Then, enter a list of permissions to turn on or off.

You turn on a permission by giving its abbreviation preceded by a hyphen (-). You turn it off by preceding its abbreviation with a hyphen followed by the letter n (-n). Permissions not explicitly named are unaffected.

If no permissions are specified on the command line, the current file attributes are displayed.

You cannot examine or change the attributes of a file you do not own unless you are the super user. A super user can examine or change the attributes of any file in the system.

The file permission abbreviations are:

d	=	Directory file
s	=	Single user file. s denotes a non-sharable file.
r	=	Read permission to owner
w	=	Write permission to owner
e	=	Execute permission to owner
pr	=	Read permission to public
pw	=	Write permission to public
pe	=	Execute permission to public

NOTE: The ***owner*** is the creator of the file. Owner access is given to any user with the same ***group ID number*** as the owner. The ***public*** is any user with a different group ID number than the owner. You can determine file ownership with the dir -e command.

SPECIAL USE: You can use attr to change a directory file to a non-directory file if all entries have been deleted from it. You may also use the deldir utility to delete directory files. You cannot change a non-directory file to a directory file with this command. The directory attribute can only be turned on when a directory is created with the makdir utility.

OPTIONS:

-?	Displays the options, function, and command syntax of attr.
-a	Suppresses the printing of attributes.
-x	Searches for the specified file in the execution directory. The file must have execute permission to be found using -x.
-z	Reads the file names from standard input.
-z=<file>	Reads the file names from <file>.

EXAMPLES:

$ attr myfile	Displays the current attributes of myfile.
$ attr myfile -npr -npw	Turns off the public read and public write permissions.
$ attr myfile -rweprpwpe	Turns on both the public and owner read, write, and execute permissions.
$ attr -z	Displays the attributes of the file names read from standard input.
$ attr -z=file1	Displays the attributes of the file names read from file1.
$ attr -npwpr *	Turns off public write and turns on public read for all files in the directory.
$ attr *.lp	Lists the attributes of all files that have names ending in .lp.

backup — Make a Backup Copy of a Disk

SYNTAX: **backup [<opts>] [<srcpath> [<destpath>]]**

FUNCTION: backup physically copies all data from one device to another. A physical copy is performed sector by sector without regard to file structures. In most cases, the devices specified must have the same format and must not have defective sectors.

In the following discussions, the source disk is the disk you are backing up. The destination disk is the disk to which you are copying.

Single Drive Backup

A single drive backup requires exchanging disks in and out of the disk drive.

NOTE: Before backing up a disk, you should write protect the source disk with the appropriate write protect mechanism to prevent accidentally confusing the source disk and the destination disk during exchanges.

To begin the backup procedure, put the source disk in the drive and type backup. The system asks if you are ready to backup. Type y if you are ready.

Initially, backup reads a portion of the source disk into memory. backup then prompts you to exchange disks. Remove the source disk from the drive, and insert the destination disk. backup writes the previously stored data on to this disk. When the backup is finished, an exchange is again requested. This places the source disk back in the drive. This exchange process continues until all of the data on the disk is copied.

The -b option increases the amount of memory the backup procedure uses. This decreases the number of disk exchanges required.

Two Drive Backup

On a two drive system, the names /d0 and /d1 are assumed if both device names are omitted on the command line. If the second device name is omitted, a single unit back-up is performed on the drive specified.

To begin the backup procedure, put the source disk in the source drive and the destination disk in the destination drive. By default, the source drive is /d0 and the destination drive is /d1. Enter backup, the name of the source drive, and the name of the destination drive. The system asks if you are ready to backup. Enter y if you are ready. If no error occurs, the backup procedure is complete.

ERRORS: The backup procedure includes two passes. The first pass reads a portion of the source disk into a buffer in memory and then writes it to the destination disk. The second pass verifies that the data was copied correctly.

If an error occurs on the first pass, something is wrong with the source disk or its drive.

If an error occurs in the second pass, the problem is with the destination disk. If backup fails repeatedly on the second pass, re-format the destination disk and try to backup again.

OPTIONS:

-?	Displays the options, function, and command syntax of backup.
-b=<num>k	Allocates <num>k of memory for the backup buffer to use. backup uses a 4K buffer by default. backup runs faster if more memory is used.
-r	Causes the backup to continue if a read error occurs.
-v	Prevents backup from making a verification pass.

EXAMPLES: This example backs up the disk in /d2 to the disk in /d3:

```
$ backup /D2 /D3
```

This example backs up the disk in /d0 to the disk in /d1 without making a verification pass:

```
$ backup -v
```

This example allocates 40K of memory to use in backing up /d0 to /d2.

```
$ backup -b40 /d0 /d2
```

binex/exbin — Convert Binary Files to S-Record/S-Record to Binary

SYNTAX: **binex [<opts>] [<path1> [<path2>]]**
exbin [<path1> [<path2>]]

FUNCTION: binex converts binary files to S-record files. The exbin utility converts S-record files to binary.

S-record files are a type of text file containing records that represent binary data in hexadecimal form. This Motorola-standard format is often directly accepted by commercial PROM programmers, emulators, logic analyzers, and similar devices that use the RS-232 interface. It can be useful for transmitting files over data links that can only handle character type data. It can also be used for converting OS-9 assembler or compiler generated programs to load on non-OS-9 systems.

binex converts the OS-9 binary file specified by <path1> to a new file with S-record format. The new file is specified by <path2>. S-records have a header record to store the program name for informational purposes and each data record has an absolute memory address. This absolute memory address is meaningless to OS-9 because OS-9 uses position-independent code.

binex currently generates the following S-record types:

S1 records	Use a two byte address field
S2 records	Use a three byte address field
S3 records	Use a four byte address field
S7 records	Terminate blocks of S3 records
S8 records	Terminate blocks of S2 records
S9 records	Terminate blocks of S1 records

To specify the type of S-record file to generate, use the -s=<num> option. <num> = 1, 2, etc., corresponding to S1, S2, etc.

exbin is the inverse operation. <path1> is assumed to be an S-Record format text file which exbin converts to pure binary form in a new file, <path2>. The load addresses of each data record must describe contiguous data in ascending order. exbin does not generate or check for the proper OS-9 module headers or CRC check value required to actually load the binary file. You can use ident to check the validity of the modules if they are to be loaded or run. exbin converts any of the S-record types mentioned above.

Using either command, standard input and output are assumed if both paths are omitted. If the second path is omitted, standard output is assumed.

OPTIONS: -? Displays the options, function, and command syntax of binex/exbin.

-a=<num> Specifies the load address in hex. This is for binex only.

-s=<num> Specifies which type of S-record format is to generate. This is for binex only.

-x binex searches for <path1> in the execution directory. This is for binex only.

EXAMPLES: The following example downloads a program to T1. This type of command downloads programs to devices such as PROM programmers.

```
$ binex scanner.S1 >/T1
```

The next example generates prog.S1 in S1 format from the binary file, prog .

```
$ binex -s1 prog prog.S1
```

The following example generates CMDS/prog in OS-9 binary format from the S1 type file, program.S1 .

```
$ exbin prog.S1 cmds/prog
```

break — Invoke Sytem Level Debugger or Reset System

SYNTAX: **break**

FUNCTION: break executes an F$SysDbg system call. This call stops OS-9 and all user processes and returns control to the ROM debugger. The debugger g[o] command resumes execution of OS-9.

You should only call break from the system's console device, because the debugger only communicates with that device. If break is invoked from another terminal, you must still use the system's console device to communicate with the debugger.

Only super users can execute break.

NOTE: break is used only for system debugging. It should not be included with or run on a production system.

NOTE: If there is no debugger in ROM or if the debugger is disabled, break will reset the system.

CAVEAT: You must be aware of any open network paths when you use the break utility as all timesharing is stopped.

OPTION: -? Displays the function and command syntax of break.

build — Build a Text File from Standard Input

SYNTAX: **build <path>**

FUNCTION: build creates a file specified by a given pathlist.

To use the build utility, type build and a pathlist. A question mark prompt (?) is displayed. This requests an input line. Each line entered is written to the output file. Entering a line consisting of only a carriage return causes build to terminate. The build utility also terminates when you enter an end-of-file character at the beginning of an input line. The end-of-file character is typically <escape>.

OPTION: -? Displays the function and command syntax of build.

EXAMPLE:

```
$ build newfile
? Build should only be used
? in creating short text files.
? [RETURN]

$ list newfile
Build should only be used
in creating short text files.
```

cfp — Command File Processor

SYNTAX: **cfp [<opts>] [<path1>] {<path2>}**

FUNCTION: cfp creates a temporary procedure file in the current data directory and then invokes the shell to execute it.

To create a temporary procedure file, type cfp, the name of the procedure file (<path1>), and the file(s) (<path2>) to be executed by the procedure file.

All occurrences of an asterisk (*) in the procedure file (<path1>) are replaced by the given pathlists, <path2>, unless preceded by the tilde character (~). For example, ~* translates to *. The command procedure is not executed until all input files have been read.

For example, if you have a procedure file in your current data directory called copyit that consists of a single command line, copy *, all of your C programs from two directories, PROGMS and MISC.JUNK, are placed in your current data directory by typing:

```
$ cfp copyit ../progms/*.c ../misc.junk/*.c
```

If you use the "-s=<string>" option, you may omit the name of the procedure file, but you must enclose the option and its string in quotes. The -s option causes the cfp utility to use the string instead of a procedure file. For example:

```
$ cfp "-s=copy *" ../progms/*.c ../misc.junk/*.c
```

NOTE: You must use double quotes to force the shell to send the string -s=copy * as a single parameter to cfp. The quotes also prevent the shell from expanding the asterisk (*) to include all pathlists in the current data directory.

In the above examples, cfp creates a temporary procedure file to copy every file ending in .c in both PROGMS and MISC.JUNK to the current data directory. The procedure file created by cfp is deleted when all the files have been copied.

Using the -s option is convenient because you do not have to edit the procedure file to change the copy procedure. For example, if you are copying large C programs, you may want to increase the memory allocation to speed up the process.

You can allocate the additional memory on the cfp command line:

```
$ cfp "-s=copy -b100 *" ../progms/*.c ../misc.junk/*.c
```

You can use the -z and -z=<file> options to read the file names from either standard input or a file. Use the -z option to read the file names from standard input. For example, if you have a procedure file called count.em that contains the command count -l * and you want to count the lines in each program to see how large the programs are before you copy them, enter the following command line:

```
$ cfp -z count.em
```

The command line prompt does not appear because cfp is waiting for input. Enter the file names on separate command lines. For example

```
$ cfp -z count.em
../progms/*.c
../misc.junk/*.c
```

When you have finished entering the file names, press the carriage return a second time to get the shell prompt.

If you have a file containing a list of the files to copy, enter:

```
$ cfp -z=files "-s=copy *"
```

OPTIONS:

-?	Displays the options, function, and command syntax of cfp.
-d	Deletes the temporary file. This is the default.
-nd	Does not delete the temporary file.
-e	Executes the procedure file. This is the default.
-ne	Does not execute the procedure file. Instead, it will dump to standard output. This option causes -d and -nd to have no effect because the temporary procedure file is not created.
-s=<str>	Reads <str> instead of a procedure file. If the string contains characters interpreted by the shell, the entire option needs to be enclosed in quotes. It does not make sense to specify both a procedure file and this option.
-t=<path>	Creates the temporary file at <path> rather than in the current working directory.
-z	Reads the file names from standard input instead of <path2>.
-z=<file>	Reads the file names from <file> instead of <path2>.

EXAMPLE: In this example, test.p is a procedure file that contains the command line list * >/p2. The command cfp test.p file1 file2 file3 produces a procedure file containing the following commands:

```
list file1 >/p2
list file2 >/p2
list file3 >/p2
```

The following command accomplishes the same thing:

```
$ cfp "-s=list * >/p2" file1 file2 file3
```

chd/chx Change Current Data Directory/Current Execution Directory

SYNTAX: **chd [<path>]**
chx <path>

FUNCTION: chd and chx are built-in shell commands used to change OS-9's working data directory or working execution directory.

To change data directories, type chd and the pathlist to the new data directory. To change execution directories, type chx and the pathlist to the new execution directory. In both cases, a full or relative pathlist may be used. Relative pathlists used by chd and chx are relative to the current data and execution directory, respectively.

If the HOME environment variable is set, the chd command with no specified directory will change your data directory to the directory specified by HOME.

NOTE: These commands do not appear in the CMDS directory as they are built-in to the shell.

EXAMPLES:

```
$ chd /d1/PROGRAMS
$ chx ..
$ chx binary_files/test_programs
$ chx /D0/CMDS; chd /D1
```

cmp — Compare Two Binary Files

SYNTAX: **cmp [<opts>] <path1> <path2>**

FUNCTION: cmp opens two files and performs a comparison of the binary values of the corresponding data bytes of the files. If any differences are encountered, the file offset (address), the hexadecimal value, and the ASCII character for each byte are displayed.

The comparison ends when an end-of-file is encountered on either file. A summary of the number of bytes compared and the number of differences found is displayed.

To execute cmp, type cmp and the pathlists of the files to be compared.

OPTIONS:

-?	Displays the options, function, and command syntax of cmp.
-b=<num>[k]	Assigns <num>k of memory for cmp to use. cmp uses a 4K memory by default.
-s	Silent mode. Stops the comparison when the first mismatch occurs and prints an error message.
-x	Searches the current execution directory for both of the specified files.

EXAMPLES: The following example uses an 8K buffer to compare file1 with file2.

```
$ cmp file1 file2 -b=8k

Differences
        (hex)  (ascii)
byte     #1  #2  #1  #2
======== ==  ==  ==  ==
00000019 72  6e   r   n
0000001a 73  61   s   a
0000001b 74  6c   t   l

Bytes compared:  0000002d
Bytes different: 00000003

file1 is longer
```

The following example compares file1 with itself.

```
$ cmp file1 file1

Bytes compared:  0000002f
Bytes different: 00000000
```

code — Print Hex Value of Input Character

SYNTAX: **code**

FUNCTION: code prints the input character followed by the hex value of the input character. Unprintable characters print as a period (.). The keys specified by tmode quit and tmode abort terminate code. tmode quit is normally <control>E, and tmode abort is normally <control>C.

The most common usage of code is to discover the value of an unknown key on the keyboard or the hex value of an ASCII character.

OPTION: -? Displays the function and command syntax of code.

EXAMPLE:

```
$ code
ABORT or QUIT characters will terminate CODE
a -> 61
e -> 65
A -> 41
. -> 10
. -> 04
$
```

compress — Compress ASCII Files

SYNTAX: **compress [<opts>] {<path>}**

FUNCTION: compress reads the specified text file(s), converts it to compressed form, and writes the compressed text file to standard output or to an optional output file.

To use compress, type compress and the path of the text file to compress. If no files are given, standard input is used.

compress replaces multiple occurrences of a character with a three character coded sequence:

aaaaabbbbbccccccccc would be replaced with **~Ea~Eb~Jc**.

Each compressed input file name is appended with _comp. If a file with this name already exists, the old file is overwritten with the new file. Typical files compress about 30% smaller than the original file.

compress reduces the size of a file to save disk space. See the expand utility for details on how to expand a compressed file.

WARNING: Only use compress and expand on text files.

OPTIONS:

-?	Displays the options, function, and command syntax of compress.
-d	Deletes the original file. This is inappropriate when no pathlist is specified on the command line and standard input is used.
-n	Creates an output file.
-z	Reads file names from standard input.
-z=<file>	Reads file names from <file>.

EXAMPLES: In the first example, file1 is compressed, file1_comp is created, and file1 is deleted.

```
$ compress file1 -dn
```

In this example, file2 is compressed, file3 is created from the redirected standard output, and file2 is deleted.

```
$ compress file2 -d >file3
```

copy — Copy Data from One File to Another

SYNTAX: **copy [<opts>] <path1> [<path2>]**

FUNCTION: copy copies data from <path1> to <path2>. If <path2> already exists, the contents of <path1> overwrites the existing file. If <path2> does not exist, it is created. If no files are given on the command line and the -z option is not specified, an error is returned.

You can copy any type of file. It is not modified in any way as it is copied. The attributes of <path1> are also copied exactly.

NOTE: You must have permission to copy the file. You must be the owner of the file specified by <path1> or have public read permission in order to copy the file. You must also be able to write to the specified directory. In either case, if the copy procedure is successful, <path2> has your group.user number unless you are the super user. If you are the super user, <path2> has the same group.user number as <path1>.

If <path2> is omitted, the destination file has the same name as the source file. It is copied into the current data directory. Consequently, the following two copy commands have the same effect:

```
$ copy /h0/cmds/file1 file1
$ copy /h0/cmds/file1
```

copy is also capable of copying one or more files to the same directory by using the -w=<dir> option. The following command copies file1 and file2 into the BACKUP directory:

```
$ copy file1 file2 -w=backup
```

If used with wildcards, the -w=<dir> option becomes a selective dsave. The following command copies all files in the current data directory that have names ending with .lp into the LP directory:

```
$ copy *.lp -w=lp
```

Data is transferred using large block reads and writes until an end-of-file occurs on the input path. Because block transfers are used, normal output processing of data does not occur on character-oriented devices such as terminals, printers, etc. Therefore, the list utility is preferred over copy when a file consisting of text is sent to a terminal or printer.

NOTE: copy always runs faster if you specify additional memory with the -b option. This allows copy to transfer data with a minimum number of I/O requests.

OPTIONS:

-?	Displays the options, function, and command syntax of copy.
-a	Aborts the copy routine if an error occurs. This option effectively cancels the continue (y/n) ? prompt of the -w option.
-b=<num>k	Allocates <num>k memory to be used by copy. copy uses a 4K memory by default.
-f	Rewrites destination files with no write permission.
-p	Does not print a list of the files copied. This option is only for copying multiple files.
-r	Overwrites the existing file.
-v	Verifies the integrity of the new file.
-w=<dir>	Copies one or more files to <dir>. This option prints the file name after each successful copy. If an error such as no permission to copy occurs, the prompt continue (y/n) ? is displayed.
-x	Uses the current execution directory for <path1>.
-z	Reads file names from standard input.
-z=<file>	Reads file names from <file>.

EXAMPLES: The following example copies file1 to file2. If file2 already exists, error #218 is returned.

```
$ copy file1 file2
```

This example copies file1 to file2 using a 15K buffer.

```
$ copy file1 file2 -b=15k
```

This example copies all files in the current data directory to MYFILE.

```
$ copy * -w=MYFILE
```

This example copies all files in the current data directory that have names ending in .lp.

```
$ copy *.lp -w=MYFILE
```

This example copies /d1/joe and /d0/jim to FILE.

```
$ copy /d1/joe /d0/jim -w=FILE
```

This example writes file3 over file4.

```
$ copy file3 file4 -r
```

count — Count Characters, Words, and Lines in a File

SYNTAX: **count [<opts>] {<path>}**

FUNCTION: count counts the number of characters in a file and optionally prints a breakdown consisting of each unique character found and the number of times it occurred.

To count the number of characters in a file, enter count and the pathlist of the file to examine. If no pathlist is specified, count examines lines from standard input.

count recognizes the tab, line feed, and form feed characters as line delimiters.

By using the -w option, count counts the number of words in a file. A word is defined as a sequence of nonblank, non-carriage-return characters.

By using the -l option, the number of lines in a file is displayed. A line is defined by zero or more characters ending in a carriage-return.

OPTIONS:

-?	Displays the options, function, and command syntax of count.
-b	Counts characters and gives a breakdown of their occurrence.
-c	Counts characters.
-l	Counts lines.
-w	Counts words.
-z	Reads file names from standard input.
-z=<file>	Reads file names from <file>.

EXAMPLE:

```
$ list file1
first line
second line
third line

$ count -clw file1
"file1" contains 34 characters
"file1" contains 6 words
"file1" contains 3 lines
```

date — Display System Date and Time

SYNTAX: **date [<opts>]**

FUNCTION: date displays the current system date and system time. The system date and time are set by the setime utility.

OPTIONS:

-? Displays the options, function, and command syntax of date.

-j Displays the Julian date and time.

-m Displays the military time (24 hour clock) after the date.

EXAMPLES:

$ date

December 18, 1990 Tuesday 2:20:20 pm

$ date -m

December 18, 1990 Tuesday 14:20:24

The following example redirects the current date and time to the printer:

$ date >/p

dcheck — Check the Disk File Structure

SYNTAX: **dcheck [<opts>] <devname>**

FUNCTION: dcheck is a diagnostic tool used to detect the condition and the general integrity of the directory/file linkages of a disk device.

To use dcheck, type dcheck, the option(s) desired, and the name of the disk device to check.

dcheck first verifies and prints some of the vital file structure parameters. It moves down the tree file system to all directories and files on the disk. As it moves down the tree file system, the integrity of the file descriptor sectors (FDs) is verified. Any discrepancies in the directory/file linkages are reported.

From the segment list associated with each file, dcheck builds a sector allocation map. This map is created in memory.

If any FDs describe a segment with a cluster not within the file structure of the disk, a message is reported:

***** Bad FD segment (xxxxxx-yyyyyy)**

This indicates that a segment starting at sector xxxxxx (hexadecimal) and ending at sector yyyyyy cannot be used on this disk. The entire FD is probably bad if any of its segment descriptors are bad. Therefore, the allocation map is not updated for bad FDs.

While building the allocation map, dcheck ensures that each disk cluster appears only once in the file structure. If a cluster appears more than once, dcheck displays a message:

Sector xxxxxx (byte=nn bit=n) previously allocated

This message indicates the cluster at sector xxxxxx has been found at least once before in the file structure. byte=nn bit=n specifies in which byte of the bitmap this error occurred and in which bit in that byte. The first byte in the bitmap is numbered zero. For dcheck's purposes, bits are numbered zero through seven; the most significant bit is numbered zero. The message may be printed more than once if a cluster appears in a segment in more than one file.

Occasionally, sectors on a disk are marked as allocated even though they are not associated with a file or the disk's free space. This is most commonly caused by media defects discovered by format. These defective sectors are not included in the free space for the disk. This can also happen if a disk is removed from a drive while files are still open, or if a directory containing files is deleted by a means other than deldir.

If all the sectors of a cluster are not used in the file system, dcheck prints a message:

xxxxxx cluster only partially used

The allocation map created by dcheck is then compared to the allocation map stored on the disk. Any differences are reported in messages:

Sector xxxxxx (byte=nn bit=n) not in file structure
Sector xxxxxx (byte=nn bit=n) not in bit map

The first message indicates sector number xxxxxx was not found as part of the file system but is marked as allocated in the disk's allocation map. In addition to the causes previously mentioned, some sectors may have been excluded from the allocation map by the format program because they were defective. They could be the last sectors of the disk, whose sum is too small to comprise a cluster.

The second message indicates that the cluster starting at sector xxxxxx is part of the file structure but is not marked as allocated in the disk's allocation map. This type of disk error could cause problems later. It is possible that this cluster may later be allocated to another file. This would overwrite the current contents of the cluster with data from the newly allocated file. All current data located in this cluster would be lost. Any clusters reported as previously allocated by dcheck have this problem.

Repairing the Bitmap

dcheck is capable of repairing two types of disk problems using the -r and the -y options. If a cluster was found in the file structure but not in the bitmap, the bit may be turned on in the bitmap to include the cluster. If the cluster was marked in the bitmap but not in the file structure, the bit in the bitmap may be turned off.

WARNING: Do not use either of these options unless you thoroughly understand what you are doing. These errors could be caused by previously mentioned problems and perhaps should not be repaired.

Restrictions:

- ¿ Only the super user (user 0.n) may use this utility.
- ¡ dcheck should have exclusive access to the disk being checked. dcheck can be fooled if the disk allocation map changes while it is building its bitmap file from the changing file structure.

OPTIONS:

-?	Displays the options, function, and command syntax of dcheck.
-d=<num>	Prints the path to the directory <num> deep.
-r	Repair mode. Prompts to turn on or off bits in the bit map.
-y	Repair mode. Does not prompt for repair, but answers yes to all

prompts. This option must be used with the -r option.

EXAMPLE:

```
$ dcheck /d2
Volume - 'Ram Disk (Caution: Volatile)' on device /dd
$001000 total sectors on media, 256 bytes per sector
Sector $000001 is start of bitmap
$0200 bytes in allocation map, 1 sector(s) per cluster
Sector $000003 is start of root dir
Building allocation map...
$0003 sectors used for id sector and allocation map
Checking allocation map...

'Ram Disk (Caution: Volatile)' file structure is intact
5 directories, 60 files
580096 of 1048576 bytes (0.55 of 1.00 meg) used on media
```

deiniz — Detach a Device

SYNTAX: **deiniz [<opts>] {<modname>}**

FUNCTION: When a device is no longer needed, use deiniz to remove the device from the system device table. deiniz uses the I$Detach system call to accomplish this. Information concerning I$Detach is located in the ***OS-9 Technical Manual***.

To remove a device from the system device table, type deiniz, followed by the name of the module(s) to detach. <modname> may begin with a slash (/). The module names may be read from standard input or from a specified pathlist if the -z option is used.

WARNING: Do not deiniz a module unless you have explicitly iniz-ed it. If you do deiniz a device that you have not iniz-ed, you could cause problems for other users who may be using the module.

OPTIONS:

-?	Displays the options, function, and command syntax of deiniz.
-z	Reads the module names from standard input.
-z=<file>	Reads the module names from <file>.

EXAMPLE: **$ deiniz t1 t2 t3**

del — Delete a File

SYNTAX: **del [<opts>] {<path>}**

FUNCTION: del deletes the file(s) specified by the pathlist(s). You must have write permission for the file(s) to be deleted. You cannot delete directory files with this utility unless their attribute is changed to non-directory.

OPTIONS:

-?	Displays the options, function, and command syntax of del.
-e	Erases the disk space that the file occupied.
-f	Delete files with no write permission.
-p	Prompts for each file to be deleted with the following prompt: **delete <filename> ? (y,n,a,q)** y = yes. n = no. a = delete all specified files without further prompts. q = quit the deleting process.
-x	Looks for the file in the current execution directory.
-z	Reads the file names from standard input.
-z=<file>	Reads the file names from <file>.

EXAMPLES: These examples use the following directory structure:

```
$ dir
  Directory of /D1  14:29:46
junk          myfile        newfile       number_five
old_test_program            test_program
```

$ del newfile Deletes newfile.

$ del *_* Deletes all files in the current data directory with an underscore character in their name.

After executing the preceding two examples, the directory has the following files:

```
$ dir
  Directory of /D1  14:30:37
junk          myfile
```

To delete all files in the current directory, type:

```
$ dir -u ! del -z
```

SEE ALSO: The attr and deldir utility descriptions

deldir — Delete All Files in a Directory

SYNTAX: **deldir [<opts>] {<path>}**

FUNCTION: deldir deletes directories and the files they contain one file at a time. deldir is only used to delete all files in the directory.

When deldir is run, it prints a prompt message:

```
$ deldir OLDFILES

Deleting directory: OLDFILES
Delete, List, or Quit (d, l, or q) ?
```

A d response initiates the process of deleting files. An l response causes dir -e to run so you can have an opportunity to see the files in the directory before they are deleted. A q response aborts the command before action is taken. After listing the files, deldir prompts with:

```
delete ? (y,n)
```

The directory to be deleted may include directory files, which may themselves include directory files, etc. In this case, deldir operates recursively (that is, lower-level directories are also deleted). The lower-level directories are processed first.

You must have correct access permission to delete all files and directories encountered. If not, deldir aborts upon encountering the first file for which you do not have write permission.

deldir automatically calls dir and attr, so they must reside in the current execution directory. When deldir calls dir, it executes a dir -a command to show all files contained in the directory.

NOTE: You should never delete the current data directory (.).

OPTIONS:

-?	Displays the options, function, and command syntax of deldir.
-f	Deletes files regardless of whether write permission is set.
-q	Quiet mode. No questions are asked. The directory and its subdirectories are all deleted, if possible.
-z	Reads the file names from standard input.
-z=<file>	Reads the file names from <file>.

devs — Display System's Device Table

SYNTAX: **devs**

FUNCTION: devs displays a list of the system's device table. The device table contains an entry for each active device known to OS-9. devs does not display information for uninitialized devices.

The devs display header lists the system name, the OS-9 version number, and the maximum number of devices allowed in the device table.

Each line in the devs display contains five fields:

Name	Description
Device	Name of the device descriptor
Driver	Name of the device driver
File Mgr	Name of the file manager
Data Ptr	Address of the device driver's static storage
Links	Device use count

NOTE: Each time a user executes a chd to an RBF device, the use count of that device is incremented by one. Consequently, the Links field may be artificially high.

OPTION: -? Displays the function and command syntax of devs.

EXAMPLE: The following example displays the device table for a system named Tazz:

```
$ devs
TAZZ_VME147  OS-9/68030 V2.4.x82  (128 devices max)

 Device    Driver    File Mgr   Data Ptr  Links
---------- ---------- ---------- --------- ------
term      sc8x30    scf       $007fda40     7
h0        rbsccs    rbf       $007fcbe0  31175
d0        rb320     rbf       $007e94a0     1
dd        rbsccs    rbf       $007fcbe0    23
t10       sc335     scf       $006d3a70     5
t11       sc335     scf       $006d3850     5
t12       sc335     scf       $006d3630     5
t13       sc335     scf       $006d3410     5
t20       sc335     scf       $006d31f0     5
t21       sc335     scf       $006d2fd0     5
t22       sc335     scf       $006d2db0     5
t23       sc335     scf       $006d2b90     5
5803      rb320     rbf       $007e94a0    20
3803      rb320     rbf       $007e94a0     1
mt2       sbgiga    sbf       $006d9640     1
n0        n9026     nfm       $006d63a0   372
nil       null      scf       $006d6340    10
socket    sockdvr   sockman   $006c0500     4
lo0       ifloop    ifman     $006c0380     4
le0       am7990    ifman     $006bed60     1
pipe      null      pipeman   $0068ecc0     3
pk        pkdvr     pkman     $0048dc90     1
pkm00     pkdvr     pkman     $00427b50     1
3807      rb320     rbf       $007e94a0    12
pcd0      rb320     pcf       $007e94a0     3
pks00     pkdvr     scf       $004279b0     2
```

SEE ALSO: The iniz and deiniz utilities

dir — Display Names of Files in a Directory

SYNTAX: **dir [<opts>] {<path>}**

FUNCTION: dir displays a formatted list of file names of the specified directory file on standard output.

To use the dir utility, type dir and the directory pathlist, if desired. If no parameters are specified, the current data directory is shown. If you use the -x option, the current execution directory is shown. If a pathlist of a directory file is specified, the files of the indicated directory are shown.

If the -e option is included, each file's entire description is displayed: size, address, owner, permissions, date, and time of last modification. Because the shell does not interpret the -x option, wildcards do not work as expected when this option is used.

Unless the -a option is used, file names that begin with a period (.) are not displayed.

Unformatted Directory Listing

You can print an unformatted directory listing using the -u option. This allows only the names of the entries of a directory to be displayed. No directory header is displayed. Entries are printed as follows:

```
$ dir -u
file1
file2
file3
DIR1
```

The output of a dir -u can be sent through a pipe to another utility or program that can use a pipe. For example:

```
$ dir -u ! attr -z
```

This displays the attributes of every entry in the current directory.

The -e option can be used to display an extended directory listing without the header by adding the -u option.

OPTIONS:

-?	Displays the options, function, and command syntax of dir.
-a	Displays all file names in the directory. This includes file names beginning with a period.
-d	Appends a slash (/) to all directory names listed. This does not affect the actual name of the directory.
-e	Displays an extended directory listing excluding file names beginning with a period.
-n	Displays directory names without displaying the file names they contain. This option is especially useful with wildcards.
-r	Recursively displays the directories. This does not include file names beginning with a period.
-r=<num>	Displays the directories recursively up to the <num> level below the current directory. This does not include file names beginning with a period.
-s	Displays an unsorted listing. This does not include file names beginning with a period.
-u	Displays an unformatted listing. This does not include file names beginning with a period.
-x	Displays the current execution directory. This does not include file names beginning with a period.
-z	Reads the directory names from standard input.
-z=<file>	Reads the directory names from <file>.

EXAMPLES: The first example displays the current data directory:

```
$ dir

                    Directory of . 12:12:54
BK          BKII        RELS        ed10.c      ed11.c
ed2.c
```

In the second example, the parent of the working data directory is displayed:

```
$ dir ..
```

This example displays the NEWSTUFF directory:

```
$ dir NEWSTUFF
```

The next example displays the entire description of the current data directory:

```
dir -e
                Directory of . 13:54:44
 Owner    Last modified Attributes Sector Bytecount Name
-------   ------------- ---------- ------ --------- ----
  1.78    90/11/28 0357  d-ewrewr   383C8       160 NOTES
  1.78    90/11/28 0357  d-ewrewr   383E8       608 PROGRAMS
  1.78    90/11/28 0357  d-ewrewr   383D8       160 TEXT
  1.78    90/11/14 0841  ------wr    F4058       438 arrayex.c
  1.78    90/11/12 0859  ------wr    F4068       538 arrayex.r
  1.78    90/11/09 0852  ------wr    F2AB0       312 asciiinfo
  0.0     90/04/27 1719  ----r-wr   71EC8      4626 atari.doc
  1.78    90/11/14 0911  ------wr    B4548       636 bobble.c
  1.78    90/11/14 0910  ------wr    B4AA8       815 bobble.r
  1.78    90/10/18 1259  ------wr    BD418       619 cd.order
  1.78    90/06/06 1009  ---wr-wr     82B8      5420 cdichanges
  1.78    90/11/28 1102  ------wr    E0C68      1478 checks.c
  1.78    90/11/28 1102  ------wr    E1D08      1075 checks.r
  1.78    90/09/07 0848  ------wr    708B8       274 datafile
  0.78    90/04/12 1206  ---wr-wr    70EE8      1065 drvr.a
  1.78    90/11/13 1544  ------wr    B1650       112 exloop
```

To display the execution directory, type:

```
$ dir -x
```

To display the entire description of the execution directory, type:

```
$ dir -xe
```

To display the contents of the current directory and all directories one level below this directory, type:

```
$ dir -r=1
```

The next example displays the entire description of all files within the current directory. This includes files within all subdirectories of the current directory.

```
$ dir -er
```

This example displays all directory and file names that begin with B.

```
$ Dir -n B*
```

diskcache — Enable, Disable, or Display Status of Cache

SYNTAX: **diskcache [<opts>] [<dev>]**

FUNCTION: diskcache enables, disables, or displays the status of the cache. Caching may be enabled for any type of RBF device, and more than one device may be cached at a time.

The total amount of system memory used for caching all enabled drives can be set by the utility's -t option. If not explicitly defined, the diskcache utility automatically selects a reasonable value based upon the amount of free system memory.

Caching may be dynamically enabled or disabled on a per drive basis while the system is running using the -e and -d options.

Statistical information regarding the hit/miss ratios, amount of memory allocated, etc. can be inspected on a drive by drive basis using the -l option. An example output of this information follows:

```
Current size =  1047552
  Size limit =  1048576

Device:  /h0:1:1
          Requests    Sectors      Hits      Zaps     >2 Xfr Hit Rate
    Reads:     47592      55436     21874       143       662  39.5%
   Writes:      7723       8065              7342        68
 Dir Reads:     54048      54048     34526     18387<-Sctr Zero  63.9%
Dir Writes:         0          0
 Hit compares =      63399 (         1/hit)
Miss compares =      92685 (         3/miss)
```

CAVEATS: Caching should only be invoked on devices that are known to the I/O system (that is, the devices should have been initialized with the iniz utility).

If caching is to be enabled on drives with different sector sizes, the device with the largest sector size should be included in the initial cache enabling. Attempting to add a drive (with a sector size larger than any currently cached drive) to the cache system after initial cache startup results in continuous "misses" for that drive, as the sector size is too large.

OPTIONS:

-?	Displays the options, function, and command syntax of diskcache.
-d	Disables cache for <dev>.
-e	Enables cache for <dev>.
-l	Displays the cache status for <dev>.
-t=<size>[k]	Specifies the size limit of the total cache.

dsave — Generate Procedure File to Copy Files

SYNTAX: **dsave [<opts>] [<path>]**

FUNCTION: dsave is used to backup or copy all files in one or more directories. It generates a procedure file, which is either executed later to actually do the work or is executed immediately using the -e option.

To use dsave, type dsave and the path of the new directory. When dsave is executed, it writes commands on standard output to copy files from the current data directory to the directory specified by <path>. If no <path> is specified, the copies are directed to the current data directory when the procedure file is executed.

dsave's standard output should be redirected to a procedure file that can be executed at a later time or the -e option should be used to execute dsave's output immediately.

If dsave encounters a directory file, it automatically includes makdir and chd commands in the output before generating copy commands for files in the subdirectory. The procedure file duplicates all levels of the file system connected downward from the current data directory.

If the current working directory happens to be the root directory of the disk, dsave creates a procedure file to backup the entire disk, file by file. This is useful when you need to copy many files from different format disks, or from a floppy disk or a hard disk.

If an error occurs, the following prompt is displayed:

continue (y,n,a,q)?

A y indicates you want to continue. An n indicates you do not want to continue. An a indicates you want to copy all possible files and you do not want dsave to display the prompt on error. A q indicates you want to quit the dsave procedure. If for any reason you do not wish to be bothered by this prompt, the -s option is available. This skips any file which cannot be copied and continues the dsave routine with no prompt.

dsave helps keep up-to-date directory backups. When the -d or -d=<date> options are used, dsave compares the date of the file to copy with a file of the same name in the directory it is to be copied to. The -d option copies any file with a more recent date. To copy a file with a date more recent than that specified, use the -d=<date> option.

A common error occurs when using dsave if the destination directory has files with the same name as the source directory. Because a file name must be unique within a directory, this produces an error. Use the -r option to prevent this error.

OPTIONS:

-? Displays the options, function, and command syntax of dsave.

-a	Does not copy any file that has a name beginning with a period.
-b[=]<n>k	Allocates <n>k bytes of memory for copy and cmp if needed.
-d	Compares dates with files of the same name and copies files with more recent dates.
-d=<date>	Compares the specified date with the date of files with the same name and copies any file with a more recent date than that specified.
-e	Executes the output immediately.
-f	Uses copy's -f option to force the writing of files.
-i	Indents for directory levels.
-l	Does not save directories below the current level.
-m	Does not include makdir commands in the procedure file.
-n	Does not load copy or cmp if -v is specified.
-o	Uses os9gen to create a bootfile on the specified destination device if a bootfile exists on the source device. The default name used for the bootfile is OS9Boot. This option is used to create a bootable disk. Merely copying OS9Boot to a new disk does not make it bootable.
-o=<name>	Uses os9gen to create a bootfile on a new device, using the specified name. This option is used to create a bootable disk. Merely copying OS9Boot to a new disk does not make it bootable.
-r	Writes any source file over a file with the same name in the destination directory. Effectively, this uses the copy utility with the -r option.
-s	Skips files on error. This effectively turns off the prompt to continue the dsave routine when an error occurs.
-v	Verifies files with the cmp utility.

EXAMPLES: The first three examples effectively accomplish the same goal: copying all files in /d0/MYFILES/STUFF to /d1/BACKUP/STUFF. Each example highlights a different method of using dsave.

In the first example, no path is specified in the dsave command and a procedure file is generated. Therefore, you must change data directories before executing the procedure file. If the directory is not changed, an error message occurs: #218--file already exists in this directory under the same name.

$ chd /d0/MYFILES/STUFF	*Selects the directory to be copied.*
$ dsave >/d0/makecopy	*Makes the procedure file* makecopy.
$ chd /d1/BACKUP/STUFF	*Select the destination directory for* makecopy.
$ /d0/makecopy	*Runs* makecopy.

The second example uses the path /d1/BACKUP/STUFF in the dsave command. Consequently, you do not need to change directories before executing the procedure file. This example also allocates 32K of memory for the copy procedure. Allocating more memory for the copy procedure usually saves time.

```
$ chd /d0/MYFILES/STUFF
$ dsave -ib=32  /d1/BACKUP/STUFF >saver
$ saver
```

The third example effectively accomplishes the same thing, but without using a procedure file.

```
$ chd /d0/MYFILES/STUFF
$ dsave -ieb32 /d1/BACKUP/STUFF
```

In the following example, dir -e shows the creation dates of the files. This shows the -d option of dsave.

```
$ chd /d0/BACKUP
$ dir -e
  Directory of . 14:14:32
 Owner    Last Modified  Attributes Sector Bytecount Name
-------   -------------  ---------- ------ --------- ----
 12.4     90/12/01 1417   ------wr    1A2B     11113 program.c
 12.4     90/06/05 1601   ------wr    8543      5744 prog.2
$ chd /d0/WORKFILES
$ dir -e
    Directory of . 14:14:32
 Owner    Last Modified  Attributes Sector Bytecount Name
-------   -------------  ---------- ------ --------- ----
 12.4     90/12/01 1417   ------wr    DODO     11113 program.c
 12.4     90/12/01 1601   ------wr    3458      5780 prog.2
directory of . 14:14:40
$ dsave -deb32 /d0/BACKUP
$ chd /d0/BACKUP
$ dir -e
 Directory of . 14:14:32
 Owner    Last Modified  Attributes Sector Bytecount Name
-------   -------------  ---------- ------ --------- ----
 12.4     90/12/01 1417   ------wr    DD33     11113 program.c
 12.4     90/12/01 1601   ------wr    4356      5780 prog.2
```

In this example only prog2 was copied because the date was more recent in the WORKFILE directory.

dump — Formatted File Data Dump in Hexadecimal and ASCII

SYNTAX: **dump [<opts>] [<path> [<addr>]]**

FUNCTION: dump produces a formatted display of the physical data contents of <path>. <path> may be a mass storage file or any other I/O device. dump is commonly used to examine the contents of non-text files.

To use this utility, type dump and the pathlist of the file to display. An address within a file may also be displayed. If <path> is omitted, standard input is used. The output is written to standard output. When <addr> is specified, the contents of the file are displayed starting with the appropriate address. <addr> is presumed to be a hexadecimal number.

The data is displayed 16 bytes per line in both hexadecimal and ASCII character format. Data bytes that have non-displayable values are represented by periods in the character area.

The addresses displayed on the dump are relative to the beginning of the file. Because memory modules are position-independent and stored in files exactly as they exist in memory, the addresses shown on the dump are relative to the load addresses of the memory modules.

OPTIONS:

-? Displays the options, function, and command syntax of dump.

-c Does not compress duplicate lines.

-m Dumps from a memory resident module.

-s Interprets the starting offset as a sector number. This is useful for RBF devices with a sector size not equal to 256.

-x Indicates that <path> is an execution directory. You must have execute permission for the pathlist.

EXAMPLES:

$ dump	Displays keyboard input in hex.
$ dump myfile >/P	Dumps myfile to printer.
$ dump shortfile	Dumps shortfile.

SAMPLE OUTPUT:

(starting address) **(data bytes in hexadecimal format)** **(data bytes in ASCII format)**

```
 Addr    0 1  2 3  4 5  6 7  8 9  A B  C D  E F 0 2 4 6 8 A C E
--------  ---- ---- ---- ---- ---- ---- ---- ---- ----------------
00000000  6d61 696e 2829 0d7b 0d09 696e 7420 783b main().{..int x;
00000010  0d09 0d09 6765 745f 7465 726d 5f64 6566 ....get_term_def
00000020  7328 293b 0d09 783d 6d65 6e75 2829 3b0d s();..x=menu();.
```

echo — Echo Text to Output Path

SYNTAX: **echo [<opts>] {<text>}**

FUNCTION: echo echoes its parameter to the standard output path. echo is typically used to generate messages in shell procedure files or to send an initialization character sequence to a terminal.

To use the echo utility, type echo and the text to output. echo reads the text until a carriage return is encountered. The input then echoes on the output path.

A hexadecimal number representing a character may be imbedded in a character string but you must precede it with a backslash (\). The shell removes all but one imbedded space from character strings passed to echo. Therefore, to allow for more than one blank between characters, you must enclose the string with double quotes. A single backslash (\) is echoed by entering two backslashes (\\).

NOTE: Do not include any of the punctuation characters used by the shell in the text unless you enclose the string with double quotes.

OPTIONS:

-?	Displays the options, function, and command syntax of echo.
-n	Separates the text with carriage returns.
-r	Does not send a carriage return after <text>.
-z	Reads the text from standard input.
-z=<file>	Reads the text from <file>.

EXAMPLES:

```
$ echo "Here is an important message!"
Here is an important message!
```

```
$ echo \1b >/p1
```
Sends an <escape> character to a printer (/p1).

```
$ echo column1      column2      column3
column1 column2 column3
```

```
$ echo "column1      column2      column3"
column1      column2      column3
```

edt — Line-Oriented Text Editor

SYNTAX: **edt [<opts>] <path>**

FUNCTION: edt is a line-oriented text editor that allows you to create and edit source files.

To use the line-oriented text editor, type edt and the pathlist desired. If the file is new or cannot be found, edt creates and opens it. edt then displays a question mark prompt (?) and waits for a command. If the file is found, edt opens it, displays the last line, and then displays the ? prompt.

The first character of a line must be a space if text is to be inserted. If any other character is typed in the first character position, edt tries to process the character as an edt command. edt command format is very similar to BASIC's editor.

edt determines the size of the file to edit and uses the returned size plus 2K as the edit buffer. If the file does not already exist, the edit buffer is initialized to 2K. When the end of the edit buffer is reached, a message is displayed.

OPTIONS:

-?	Displays the options, function, and command syntax for edt.
-b=<num>k	Allocates a buffer area equal to the size of the file plus <num>k bytes. If the file does not exist, a buffer of the indicated size is assigned for the new file.

EDT COMMANDS: All edt commands begin in the first character position of a line.

<num>	Moves the cursor to line number <num>.
<esc>	Closes the file and exits. q also does this.
<cr>	Moves the cursor down one line (carriage return).
+<num>	Moves the cursor down <num> lines. Default is one.
-<num>	Moves the cursor up <num> lines. Default is one.
<space>	Inserts lines.
d[<num>]	Deletes <num> lines. If <num> is not specified, the default value of <num> is one.
l<num>	Lists <num> lines. <num> may be positive or negative. The default value of <num> is one.
l*	Lists all lines in the entire file.
q	Quits the editing session. Command returns to the program that called the editor or the OS-9 shell.

NOTE: For the following search and replace commands, <delim> may be any character. The asterisk (*) option indicates that all occurrences of the pattern are searched for and replaced if specified.

s[*]<delim><search string><delim>

Search command: searches for the occurrences of a pattern. For example:

s/and/	Finds the first occurrence of and.
s*,Bob,	Finds all occurrences of Bob.

c[*]<delim><search string><delim><replace string><delim>

Replace command: finds and replaces a given string. For example:

c/Tuesday/Wednesday/	Replaces the first occurrence of Tuesday with Wednesday.
c*"employee"employees"	Replaces all occurrences of employee with employees.

events — Display Active System Events

SYNTAX: **events**

FUNCTION: events displays a list of the active events on the system and information about each event. The events header line lists the system name and the OS-9 version number.

Each line in the events display contains six fields:

event ID	Event ID number
name	Name of the event
value	Current contents of the event variable
W-inc	Wait increment. Assigned when the event is created and does not change.
S-inc	Signal increment. Assigned when the event is created and does not change.
links	Event use count. When the event is created, links is assigned the value one. links is incremented each time a process links to the event.

An event cannot be deleted unless the link count is zero.

If no active events are currently on the system, events displays the message "No active events."

OPTION: -? Displays the function and command syntax of events.

EXAMPLE: The following example displays the active system events for a system named Calvin:

```
Calvin    OS-9/68K V2.4

event ID      name        value     W-inc  S-inc  links
---------  ------------ ---------- ------ ------ ------
  10000   evtfffe4000          1     -1      1      1
  20001   irqfffe4000          0     -1      1      1
  30002   SysMbuf         121952      0      0      1
  40003   net_input            0     -1     -1      1
  50004   Sur00227750          0      0      0      1
  60005   Str002261f0          0      0      0      1
  70006   Stw002261f0          0      0      0      1
  80007   Str00227380          0      0      0      1
  90008   Stw00227380          0      0      0      1
  a0009   Str00232a50          0      0      0      1
  b000a   Stw00232a50          0      0      0      1
  c000b   Str0020ac30          0      0      0      1
  d000c   Stw0020ac30          0      0      0      1
  e000d   pkm00i               0      0      0      1
  f000e   pkm00o               0      0      0      1
 10000f   teln.1               0     -1     -1      1
 130012   Str0020adf0          0      0      0      1
 140013   Stw0020adf0          0      0      0      1
```

SEE ALSO: F$Event service request in the ***OS-9 Technical Manual***

ex — Execute Program as Overlay

SYNTAX: **ex <path> [<arglist>]**

FUNCTION: ex is a built-in shell command that causes the process executing the command to start executing another program. It permits a transition from the shell to another program without creating another process, thus conserving system memory.

ex is often used when the shell is called from another program to execute a specific program, after which the shell is not needed. For example, applications which use only BASIC need not waste memory space on shell.

ex should always be the last command on a shell input line because any command lines following it are never processed.

NOTE: Because this is a built-in shell command, it does not appear in the CMDS directory.

EXAMPLES:

```
$ ex BASIC
$ tsmon /t1& tsmon /t2& ex tsmon /term
```

expand — Expand a Compressed File

SYNTAX: **expand [<opts>] {<path>}**

FUNCTION: expand restores compressed files to their original form. It is the complement command of the compress utility.

To expand a compressed file, type expand and the name of the file to expand. If no file names are given on the command line, standard input is assumed.

OPTIONS:

-?	Displays the options, function, and command syntax of expand.
-d	Deletes the old version of the file. This option is not appropriate when no pathlist is specified on the command line and standard input is used.
-n	Sends output to a file instead of the standard output. The file has _exp appended to it, unless the file name already has a _comp suffix. In this case, the _comp is removed.
-z	Reads the file names from standard input.
-z=<file>	Reads the file names from <file>.

EXAMPLES:

$ expand data.a -nd	Expands and then deletes data.a, creating data.a_exp.
$ expand file1_comp	Expands file1_comp and displays output on standard output.
$ expand -nd file2_comp	file2_comp is expanded and then deleted, creating file2 with the expanded output.

fixmod — Fix Module CRC and Parity

SYNTAX: **fixmod [<opts>] {<modname>}**

FUNCTION: fixmod verifies and updates module parity and module CRC (cyclic redundancy check). You can also use it to set the access permissions and the group.user number of the owner of the module.

Use fixmod to update the CRC and parity of a module every time a module is ***patched*** or modified in any way. OS-9 cannot recognize a module with an incorrect CRC.

You must have write access to the file in order to use fixmod.

Use the -u option to recalculate and update the CRC and parity. Without the -u option, fixmod only verifies the CRC and parity of the module.

The -up=<perm> option sets the module access permissions to <perm>. <perm> must be specified in hexadecimal. You must be the owner of the module or a super user to set the access permissions. The permission field of the module header is divided into four sections from right to left:

owner permissions
group permissions
public permissions
reserved for future use

Each of these sections are divided into four fields from right to left:

read attribute
write attribute
execute attribute
reserved for future use

The entire module access permissions field is given as a four digit hexadecimal value. For example, the command fixmod -up=555 specifies the following module access permissions field:

-----e-r-e-r-e-r

The -uo<g>.<u> option allows the super user to change the ownership of a module by setting the module owner's group.user number.

OPTIONS:	-?	Displays the options, function, and command syntax of fixmod.
	-u	Updates an invalid module CRC or parity.
	-ua[=]<att.rev>	Changes the module's attribute/revision level.
	-ub	Fixes the sys/rev field in BASIC packed subroutine modules.
	-up=<perm>	Sets the module access permissions to <perm>. <perm> must be specified in hexadecimal.
	-uo<g>.<u>	Sets the module owner's group.user number to <g>.<u>. Only the super user is allowed to use this option.
	-x	Looks for the module in the execution directory.
	-z	Reads the module names from standard input.
	-z=<file>	Reads the module names from <file>.
EXAMPLES:	**$ fixmod dt**	Checks parity and CRC for module dt.
	$ fixmod dt -u	Checks parity and CRC for module dt and updates them if necessary.

SEE ALSO: Refer to the ***OS-9 Technical Manual*** for more information concerning CRC and parity. For a full explanation of module header fields, see the ident utility.

format — Initialize Disk Media

SYNTAX: **format [<opts>] <devname>**

FUNCTION: format is used to physically initialize, verify, and establish an initial file structure on a disk. You must format all disks before using them on an OS-9 system. format can format almost any type of disk, including hard disks.

To use the format utility, type format, the name of the device to format, and any options. format will determine whether the device is autosize (for example, devices such as SCSI CCS drives) or non-autosize (such as standard floppy disks and many hard disks). An autosize device is one which can be queried to determine the capacity of the device. format checks a bit in PD_Ctrl to determine whether or not a device is autosize. If this bit is zero, the device is non-autosize. If one, the media is autosize.

Format on Non-Autosize Devices

If format determines that your device is non-autosize, format reads a description of the disk from the device descriptor module. The default values for the number of sides, number of tracks, sector size, and density are determined by the values in the descriptor. At this time, the default cluster size is set at one. format determines the media capacity by multiplying together the number of cylinders (PD_CYL), tracks (PD_TKS), and sectors per track (PD_SCT, PD_T0S). Because format calculates the device capacity in this way, the -t=<num> and -ss/-ds options can be used to affect the capacity of the device.

The following information is displayed before formatting begins:

```
            Disk Formatter
OS-9/68K V2.4   Delta MVME147 - 68030
------------  Format Data  ------------

Fixed values:
   Physical floppy size: 5 1/4"
                    (Universal Format)
          Sector size: 256
        Sectors/track: 16
    Track zero sect/trk: 16
         Sector offset: 1
          Track offset: 1
           LSN offset: $000000
Total physical cylinders: 80
 Minimum sect allocation: 8

Variables:
       Recording format: MFM all tracks
   Track density in TPI: 96
Number of log. cylinders: 79
     Number of surfaces: 2
Sector interleave offset: 1

Formatting device:  /d0
proceed?
```

You can change the values in the variables section when formatting floppy disks by command line options or by answering n to the prompt. format asks for any required options not given on the command line.

When formatting hard disks, answering n to the prompt returns control to the shell. You can change hard disk parameters only by command line options or by changing the device descriptor.

The values in the Fixed values section can only be changed by altering the device descriptor module of the specific unit.

Format on Autosize Devices

If format determines that the device has the autosize feature, format performs an SS_DSize SetStat call to the drive to request the capacity of the device. Typically, the driver then queries the actual drive. The value returned to format is the capacity of the device. Because format performs no calculations when determining the capacity, the -t and -ss/-ds options do not affect the capacity of the device.

The following information is displayed before formatting commences:

```
          Disk Formatter
OS-9/68K V2.4   Delta MVME147 - 68030
------------  Format Data  ------------

Fixed values:
          Disk type: hard
         Sector size: 512
       Disk capacity: 208936 sectors
                (106975232 bytes)
       Sector offset: 0
        Track offset: 0
          LSN offset: $000000
Minimum sect allocation: 8

Variables:
Sector interleave offset: 1

Formatting device:  /h1
proceed?
```

When formatting hard disks, answering n to the prompt returns control to the shell. You can only change the sector interleave offset. The other values cannot be changed by the format utility.

The values in the Fixed values section can only be changed by altering the device descriptor module of the specific unit.

Continuing the Format Procedure

The formatting process works as follows:

- ¿ The disk surface is physically initialized and sectored.
- ¡ Each sector is read back and verified. If the sector fails to verify after several attempts, the offending sector is excluded from the initial free space on the disk. As the verification is performed, track numbers are displayed on the standard output device for non-autosize devices; logical sector numbers are displayed for autosize devices.
- ¬ The disk allocation map, root directory, and identification sector are written to the first few sectors of track zero. These sectors cannot be defective.

NOTE: format uses a ***fast verify*** mode. This means that format reads a minimum of 32 sectors. If the cluster size is greater than 32 sectors, then one cluster worth of sectors is read. If the cluster size is less than 32 sectors, 32 sectors are read. If you want format to use the cluster size regardless of the number of sectors per cluster, you must use the -nf option. For example, if your cluster size has one sector, 32 sectors are read by default, while only one sector would be read if you specify -nf.

NOTE: You must run os9gen to create the bootstrap after the disk has been formatted if you use the disk as a system disk,

OPTIONS:

-?	Displays the options, function, and command syntax of format.
-c=<num>	Specifies the number of sectors per cluster. <num> must be decimal and must be a power of 2. The default is 1.
-dd	Double density (floppy) disk
-ds	Double sided (floppy) disk
-e	Displays elapsed verify time. This is useful for checking the sector interleave values.
-i=<num>	Specifies the number for sector interleave offset value. <num> is decimal.
-nf	Specifies no fast verify mode.
-np	Specifies no physical format.
-nv	Specifies no physical verification.
-r	Inhibits the ready prompt. This option is ignored if the device is a hard disk. This makes it necessary to explicitly state that you want to format a hard disk.
-sd	Single density (floppy) disk
-ss	Single sided (floppy) disk
-t=<num>	Specifies the number of cylinders given in decimal.
-v=<name>	Volume name. This name can be 32 characters maximum. **NOTE:** If the name contains blanks, enclose the option and name with quotation marks. For example, "-v=Name of disk".

EXAMPLES:

```
$ format /D1 -dsdd -v="database" -t=77

$ format /D1 -sssd -r
```

free — Display Free Space Remaining on a Mass-Storage Device

SYNTAX: **free [<opts>] {<devname>}**

FUNCTION: free displays the number of unused 256-byte sectors on a device available for new files or for expanding existing files. free also displays the disk's name, creation date, cluster size, and largest free block in bytes.

To use the free utility, type free followed by the name of the device to examine. The device name must be the name of a mass-storage, multi-file device.

Data sectors are allocated in groups called ***clusters***. The number of sectors per cluster depends on the storage capacity and physical characteristics of the specific device. This means that small amounts of free space, given in sectors, may not be divisible into the same number of files.

For example, a given disk system uses 8 sectors per cluster. A free command shows the disk has 32 sectors free. Because memory is allocated in clusters, a maximum of four new files could be created even if each had only one sector.

OPTIONS:

-?	Displays the option, function, and command syntax of free.
-b=<num>	Uses the specified buffer size.

EXAMPLE:

```
$ free
"Tazz: /H0 Wren V" created on: Oct 6, 1990
Capacity: 2347860 sectors (256-byte sectors, 8-sector clusters)
1508424 free sectors, largest block 1380120 sectors
386156544 of 601052160 bytes (368.26 of 573.20 Mb) free on media (64%)
353310720 bytes (336.94 Mb) in largest free block
```

frestore — Directory Backup Restoration

SYNTAX: **frestore [<opts>] [<path>]**

FUNCTION: frestore restores a directory structure from multiple volumes of tape or disk media.

Typing frestore by itself on the command line attempts to restore a directory structure from the device /mt0 to the current directory. Specifying the pathlist of a directory on the command line causes the files to be restored in the specified directory. fsave creates the directory structure and an index of the directory structure.

If more than one tape/disk is involved in the fsave backup, each tape/disk is considered a different volume. The volume count begins at one (1). When you begin a frestore operation, you must use the last volume of the backup first. The last volume of the backup contains the index of the entire backup.

frestore first attempts to locate and read in the index of the directory structure from the source device. The device you are restoring from is the source device. It then begins an interactive session with you to determine which files and directories in the backup should be restored to the current directory. The -s option forces frestore to restore all files/directories of the backup from the source device without the interactive shell.

The -d option allows you to specify a source device other than /mt0.

The -v option causes frestore to identify the name and volume number of the backup mounted on the source device. It also displays the date the backup was made and the group.user number of the person who made the backup. This option does not restore any files. After displaying the appropriate information, frestore terminates. This is helpful for locating the last volume of the backup if a mix-up has occurred. The -i option duplicates the -v option and also checks to see if the index is on the volume being checked.

The -e option echoes each file pathlist as the index is read off the source device.

CAVEATS: frestore cannot restore a file that requires more than four disks.

If the backup index requires more than a single volume, frestore fails with a header block corrupt error.

NOTE: For a full description of the fsave, frestore, and tape utilities, read the chapter on making backups. The information in the chapter on making backups includes work-through examples and backup strategies for disk and tape.

OPTIONS: -? Displays the options, function, and command syntax of frestore.

-a	Forces access permission for overwriting an existing file. You must be the owner of the file or a super user (0.n) to use this option.
-b[=]<int>	Specifies the buffer size used to restore the files.
-c	Checks the validity of files without the interactive shell.
-d[=]<path>	Specifies the source device. The default source device is /mt0.
-e	Displays the pathlists of all files in the index as the index is read from the source device.
-f[=]<path>	Restores from a file.
-i	Displays the backup name, creation date, group.user number of the owner of the backup, volume number of the disk or tape, and whether the index is on the volume. This option will not restore any files. The information is displayed, and frestore is terminated.
-j[=]<int>	Sets the minimum system memory request.
-p	Suppresses the prompt for the first volume.
-q	Overwrites already existing files when used with the -s option.
-s	Forces frestore to restore all files from the source device without an interactive shell.
-t[=]<dirpath>	Specifies an alternate location for the temporary index file.
-v	Displays the backup name, creation date, group.user number of the owner of the backup, and volume number of the disk or tape. This option will not restore any files. The information is displayed, and frestore is terminated.
-x[=]<int>	Pre-extends a temporary file. <int> is specified in kilobytes.

EXAMPLES: The following command restores files and directories from the source device /mt0 to the current directory by way of an interactive shell.

```
$ frestore
```

The next command restores files and directories from the source device /d0 to the current directory using a 32K buffer. As each file is read from the index, the file's pathlist is echoed to the terminal.

```
$ frestore -eb=32 -d=/d0
```

The next command restores all files/directories found on the source device /mt1 to the directory BACKUP without using the interactive shell.

```
$ frestore -d=/mt1 -s BACKUP
```

The following command displays the backup and the volume number:

```
$ frestore -v

Backup: DOCUMENTATION
Made:   11/30/90 10:10
By:     0.0
Volume: 0
```

This command does not restore the backup.

fsave — Incremental Directory Backup

SYNTAX: **fsave [<opts>] [<dir>]**

FUNCTION: fsave performs an incremental backup of a directory structure to tape(s) or disk(s).

Typing fsave by itself on the command line makes a level 0 backup of the current directory onto the target device /mt0.

Use the -l option to specify different backup levels. A higher level backup only saves files changed since the most recent backup with the next lower number. For example, a level 1 backup saves all files changed since the last level 0 backup.

The backup log file, /h0/sys/backup_dates, is updated each time an fsave is executed. The backup log keeps track of the name of the backup and the date it was created. More importantly, it keeps track of the level of the backup. When fsave is executed, this backup log is examined for the specified level of the current backup and the previous backups with the same name. Once the backup is finished, a new entry is entered in the file indicating the date, name, level, etc. of the current backup.

fsave does not accept a device name as a directory. For example, if fsave /ho is entered, error #216 is returned.

The Fsave Procedure

Upon starting an fsave procedure, fsave first builds the directory structure. You are then prompted to mount the first volume to use:

```
fsave: please mount volume.
(press return when mounted).
```

If a disk is used as the backup medium, fsave verifies the disk and displays the following information:

```
verifying disk

Bytes held on this disk: 546816
Total data bytes left:   62431
Number of Disks needed:  1
```

NOTE: The numbers above are used as an example. If a tape is used as the backup medium, the backup begins at this point.

As each file is saved to the backup device, its pathlist is echoed to the terminal. If this is a long backup, use the -e option to turn off the echoing of pathlists.

If fsave receives an error when trying to backup a file, it displays a message and continues the fsave operation.

If the backup requires more than one volume, fsave prompts you to mount the next volume before continuing.

At the end of the backup, fsave prints the following information:

```
fsave: Saving the index structure

Logical backup name:
Date of backup:
Backup made by:
Data bytes written:
Number of files:
Number of volumes:
Index is on volume:
```

By specifying one or more directories on the command line, fsave performs recursive backups for each specified pathlist. You can specify a maximum of 32 directories on the command line.

Use the -d option to specify an alternative target device. The default device is /mt0.

Use the -m option to specify an alternative backup log file. The default pathlist is /h0/sys/backup_dates.

WARNING: When using disks for backup purposes, be aware that fsave does not use an RBF file structure to save the files on the target disk. It creates its own file structure. This makes the backup disk unusable for purposes other than fsave and frestore without reformatting. Any data on the disk before using fsave is destroyed by the backup.

NOTE: For a full description of the fsave, frestore, and tape utilities, read the chapter on backups in this manual. The information in the chapter on backups includes work through examples and backup strategies for disk and tape.

OPTIONS:

-?	Displays the options, function, and command syntax of fsave.
-b[=]<int>	Allocates <int>k buffer size to read files from source disk.
-d[=]<dev>	Specifies the target device to store the backup. The default target device is /mt0.
-e	Does not echo the file pathlist as it is saved to the target device.
-f[=]<path>	Saves to a file.
-g[=]<int>	Specifies a backup of files owned by group number <int> only.
-j[=]<num>	Specifies the minimum system memory request.
-l[=]<int>	Specifies the level of backup to be performed.
-m[=]<path>	Specifies the pathlist of the date backup log file to be used. The default is /h0/sys/backup_dates.
-p	Turns off the ***mount volume*** prompt for the first volume.
-s	Displays the pathlists of all files needing to be saved and the size of the entire backup without actually executing the backup procedure.
-t[=]<dirpath>	Specifies an alternate location for the temporary index file.
-u[=]<int>	Specifies a backup of files owned by user number <int> only.
-v	Does not verify the disk volume when mounted.
-x[=]<int>	Pre-extends the temporary file. <int> is specified in kilobytes.

EXAMPLES: The following command specifies a level 0 backup of the current directory. It assumes the device /mt0 is to be used. /h0/SYS/backup_dates is used as the backup log file.

```
$ fsave
```

This command specifies a level 2 backup of the current directory. The device /mt1 is used. /h0/misc/my_dates is used as the backup log file.

```
$ fsave -l=2 -d=/mt1 -m=/h0/misc/my_dates
```

The next command specifies a level 0 backup of all files owned by the super user in the CMDS directory, assuming CMDS is in your current directory. /d2 is the target device used for this backup. The backup log file used is /h0/sys/backup_dates. The mount volume prompt is not generated for the first volume, and a 32K buffer is used to read the files from the CMDS directory.

```
$ fsave -pb=32 -g=0 -u=0 -d=/d2 CMDS
```

grep — Search a File for a Pattern

SYNTAX: **grep [<opts>] [<expression>] {[<path>]}**

FUNCTION: grep searches the input pathlist(s) for lines matching <expression>.

To use the grep utility, type grep, the expression to search for, and the pathlist of the file to search. If no <path> is specified, grep searches standard input.

If grep finds a line that matches <expression>, the line is written to the standard output with an optional line number of where it is located within the file. When multiple files are searched, the output has the name of the file preceding the occurrence of the matched expression.

Expressions

An <expression> is used to specify a set of characters. A string which is a member of this set is said to match the expression. To facilitate the creation of expressions, some metacharacters are defined to create complex sets of characters. These special characters are:

Char	Name/Description
.	**ANY**. The period (.) is defined to match any ASCII character except new line.
~	**BOL** or **NEGATE**. The tilde (~) is defined to modify a character class as described above when located between square brackets ([ ]). At the beginning of an entire expression, it requires the expression to compare and match the string at only the beginning of the line. The **NEGATE** character modifies the character class so it matches any ASCII character *not* in the given class or newline.
[ ]	**CHARACTER CLASS**. The square brackets ([ ]) define a group of characters which match any single character in the compare string. grep recognizes certain abbreviations to aid the entry of ranges of strings: [a-z] Equivalent to the string abcdefghijklmnopqrstuvwxyz [m-pa-f] Equivalent to the string mnopabcdef [0-7] Equivalent to the string 01234567

Char	Name/Description
*	**CLOSURE**. The asterisk (*) modifies the preceding single character expression, so it matches zero or more occurrences of the single character. If a choice is available, the longest such group is chosen.
$	**EOL**. The dollar sign ($) requires the expression to compare and match the string only when located at the end of line.
\	**ESCAPE**. The backslash (\) removes special significance from special characters. It is followed by a base and a numeric value or a special character. If no base is specified, the base for the numeric value defaults to hexadecimal. An explicit base of decimal or hexadecimal can be specified by preceding the numeric value with a qualifier of d or x, respectively. It also allows entry of some non-printing characters such as: \t = Tab character \n = New-line character \l = Line feed character \b = Backspace character \f = Form feed character

Example Expressions

You can combine any metacharacters and normal characters to create an expression:

Expression	Same as
abcd	abcd
ab.d	abcd, abxd, ab?d, etc.
"ab *d"	“abd”, “ab d”, “ab d”, “ab d”, etc.
~abcd	abcd (only if very first characters on a line)
abcd$	abcd (only if very last characters on a line)
~abcd$	abcd (only if abcd is the complete line)
[Aa]bcd	abcd, Abcd
abcd[0-9a-zA-z]	abcd followed by any alphanumeric character
bcd[~a-d]	bcd followed by any ASCII char except a, b, c, d, or new line

OPTIONS:

-?	Displays the options, function, and command syntax of grep.
-c	Counts the number of matching lines.
-e=<expr>	Searches for <expr>. This is the same as <expression> in the command line.
-f=<path>	Reads the list of expressions from <path>.
-l	Prints only the names of the files with matching lines.
-n	Prints the relative line number within the file followed by the matched expression.
-s	Silent Mode. Does not display matching lines.
-v	Prints all lines except for those that match.
-z	Reads the file names from standard input.
-z=<path>	Reads the file names from <path>.

NOTES: -l and -n cannot be used at the same time. -n and -s cannot be used at the same time.

EXAMPLES: To write all lines of myfile that contain occurrences of xyz to standard output, enter:

```
$ grep xyz myfile
```

This example searches myfile for expressions input from words, counts the number of matches, and gives the line number found with each occurrence:

```
$ grep -f=words myfile -nc
```

help On-Line Utility Reference

SYNTAX: **help [<utility name>]**

FUNCTION: help displays information about a specific utility.

For information about a specific utility, type help and the name of the desired utility. help displays the function, syntax, and options of the utility. After the information is displayed, control returns to the shell.

For information about the help utility, type help by itself. help lists the syntax and function of the help utility.

NOTE: Built-in shell commands do not have help information.

EXAMPLES: **$ help build**

$ help attr

ident — Print OS-9 Module Identification

SYNTAX: **ident [<opts>] {<modname>}**

FUNCTION: ident displays module header information and the additional information that follows the header from OS-9 memory modules. ident also checks for incomplete module headers.

ident displays the following information in this order:

module size
owner
CRC bytes (with verification)
header parity (with verification)
edition
type/language, and attributes/revision
access permission

For program modules it also includes:

execution offset
data size
stack size
initialized data offset
offset to the data reference lists

ident prints the interpretation of the type/language and attribute/revision bytes at the bottom of the display.

With the exception of the access permission data, all of the above fields are self-explanatory. The access permissions are divided into four sections from right to left:

owner permissions
group permissions
public permissions
reserved for future use

Each of these sections are divided into four fields from right to left:

read attribute
write attribute
execute attribute
reserved for future use

If the attribute is turned on, the first letter of the attribute (r, w, e) is displayed.

All reserved fields are displayed as dashes unless the fields are turned on. In that case, the fields are represented with question marks. In any case, the kernel ignores these fields as they are reserved for future use.

Owner permissions allow the owner to access the module. Group permissions allow anyone with the same group number as the owner to access the module. Public permissions allow access to the module regardless of the group.user number. The following example allows the owner and the group to read and execute the module, but bars access to the public:

Permission: $55 ---------e-r-e-r

OPTIONS:

-?	Displays the options, function, and command syntax of ident.
-m	Searches for modules in memory.
-q	Quick mode. Only one line per module.
-s	Silent mode. Quick, but only displays bad CRCs.
-x	Searches for modules in the execution directory.
-z	Reads the module names from standard input.
-z=<file>	Reads the module names from <file>.

EXAMPLE:

```
$ ident -m ident
Header for:     ident
Module size:    $1562      #5474
Owner:          0.0
Module CRC:     $FA8ECA    Good CRC
Header parity:  $2471      Good parity
Edition:        $C         #12
Ty/La At/Rev:   $101       $8001
Permission:     $555       -----e-r-e-r-e-r
Exec. off:      $4E        #78
Data size:      $15EC      #5612
Stack size:     $C00       #3072
Init. data off: $1482      #4250
Data ref. off:  $151A      #5402
Prog mod, 68000 obj, Sharable
```

iniz — Attach Devices

SYNTAX: **iniz [<opts>] {<devname>}**

FUNCTION: iniz performs an I$Attach system call on each device name passed to it. This initializes and links the device to the system.

To attach a device to the system, type iniz and the name(s) of the device(s) to be ***attached*** to the system. OS-9 searches the system module directory using the name of the device to see if the device is already attached.

If the device is not already attached, an initialization routine is called to link the device to the system.

If the device is already attached, it is not re-initialized, but the link count is incremented.

The device names may be listed on the command line, read from standard input or read from a specified pathlist.

NOTE: Do not iniz non-sharable device modules as they become "busy" forever.

OPTIONS:

-?	Displays the options, function, and command syntax of iniz.
-z	Reads the device names from standard input.
-z=<file>	Reads the device names from <file>.

EXAMPLES:

$ iniz h0 term	Increments the link counts of modules h0 and term.
$ iniz -z	Increments the link count of any modules with names read from standard input.
$ iniz -z=/h0/file	Increments the link count of all modules whose names are supplied in /h0/file.

irqs — Display System's IRQ Polling Table

SYNTAX: **irqs**

FUNCTION: irqs displays a list of the system's IRQ polling table. The IRQ polling table contains a list of the service routines for each interrupt handler known by the system.

The irqs display header lists the system name, the OS-9 version number, the maximum number of devices allowed in the device table, and the maximum number of entries in the IRQ table.

Each line in the irqs display contains seven fields:

vector	Exception vector number used by the device. A second number, the hardware interrupt level, is displayed for auto-vectored interrupts.
prior	Software polling priority.
port addr	Base address of the interrupt generating hardware. The operating system does not use this value, but passes it to the interrupt service routine.
data addr	Address of the device driver's static storage.
irq svc	Interrupt service routine's entry point.
driver	Name of the module which contains the interrupt service routine, usually a device driver.
device	Name of the device descriptor. **NOTE:** If no device name is displayed, the entries relate to IRQ handlers that support "anonymous" devices (for example, the clock ticker, DMA devices associated with other peripherals).

OPTION: -? Displays the function and command syntax of irqs.

EXAMPLE: The following example displays the IRQ polling table for a system named Calvin:

```
$ irqs

Calvin    OS-9/68K V2.4    (max devs: 32, max irqs: 32)

 vector prior port addr data addr  irq svc    driver   device
------- ----- --------- --------- ---------  --------- ------
 68        0  $fffe1800 $00230b90 $00215084  am7990
 69        5  $fffe4000 $003bd560 $00012ad2  scsi147
 70        5  $fffe4000 $003bd560 $00012ad2  scsi147
 72        0  $fffe1000 $00000000 $0000ccda  tk147
 88        5  $fffe3002 $003be3f0 $0000dacc  sc8x30    term
 88        5  $fffe3000 $003bd300 $0000dacc  sc8x30    t1
 89        5  $fffe3800 $002044a0 $0000dacc  sc8x30    t3
 89        5  $fffe3802 $003bbeb0 $0000dacc  sc8x30    t2
 90        5  $ffff1001 $003bc560 $0000e6b6  sc68560   t4
 91        5  $ffff1041 $003bc120 $0000e6b6  sc68560   t5
255        5  $ffff8800 $00245a50 $002458e0  n9026     n0
```

SEE ALSO: F$IRQ system state service request in the ***OS-9 Technical Manual***

kill — Abort a Process

SYNTAX: **kill {<procID>}**

FUNCTION: kill is a built-in shell command. It sends a signal to ***kill*** the process having the specified process ID number. This unconditionally terminates the process.

To terminate a process, type kill and the ID number(s) of the process(es) to abort. The process must have the same user ID as the user executing the command. Use procs to obtain the process ID numbers.

If a process is waiting for I/O, it cannot die until it completes the current I/O operation. Therefore, if you kill a process and procs shows it still exists, the process is probably waiting for the output buffer to be flushed before it can die.

The command kill 0 kills all processes owned by the user.

NOTE: Because kill is a built-in shell command, it does not appear in the CMDS directory.

EXAMPLES:

```
$ kill 6 7
$ procs
 Id PId Grp.Usr Prior MemSiz Sig S   CPU Time   Age Module & I/O
  2   1  0.0   128   0.25k   0 w      0.01   ??? sysgo <>>>term
  3   2  0.0   128   4.75k   0 w      4.11 01:13 shell <>>>term
  4   3  0.0     5   4.00k   0 a   12:42.06 00:14 xhog <>>>term
  5   3  0.0   128   8.50k   0 *      0.08 00:00 procs <>>term
  6   0  0.0   128   4.00k   0 s      0.02 01:12 tsmon <>>>t1
  7   0  0.0   128   4.00k   0 s      0.01 01:12 tsmon <>>>t2
$ kill 4
$ procs
 Id PId Grp.Usr Prior MemSiz Sig S   CPU Time   Age Module & I/O
  2   1  0.0   128   0.25k   0 w      0.01   ??? sysgo <>>>term
  3   2  0.0   128   4.75k   0 w      4.11 01:13 shell <>>>term
  4   3  0.0   128   8.50k   0 *      0.08 00:00 procs <>>term
```

link — Link a Previously Loaded Module into Memory

SYNTAX: **link [<opts>] {<modname>}**

FUNCTION: link is used to ***link*** a previously loaded module into memory. To use this utility, type link and the name(s) of the module(s) to lock into memory. The link count of the module specified is incremented by one each time it is linked. Use unlink to unlink the module when it is no longer needed.

OPTIONS:

-?	Displays the options, function, and command syntax of link.
-z	Reads the file names from standard input.
-z=<file>	Reads the file names from <file>.

EXAMPLES:

$ link prog1 prog2 prog3

$ link -z=linkfile	Links modules from linkfile.
$ link -z	Links modules from standard input.

list — List the Contents of a Text File

SYNTAX: **list [<opts>] {<path>}**

FUNCTION: list displays text lines from the specified path(s) to standard output.

To use the list utility, type list and the pathlist. list terminates upon reaching the end-of-file of the last input path. If more than one path is specified, the first path is copied to standard output, the second path is copied next, etc. Each path is copied to standard output in the order specified on the command line.

list is most commonly used to examine or print text files.

OPTIONS:

-?	Displays the options, function, and command syntax of list.
-z	Reads the file names from standard input.
-z=<file>	Reads the file names from <file>.

EXAMPLES: To redirect the startup listing to the printer and place the entire command in the background, enter:

```
$ list /d0/startup >/P&
```

The following example lists text from files to standard output in the same order as the command line:

```
$ list /D1/user5/document /d0/myfile /d0/Bob/text
```

To list all files in the current data directory, enter:

```
$ list *
```

The following example reads the name(s) of the file(s) to list from namefile and lists their contents.

```
$ list -z=namefile
```

load — Load Module(s) from File into Memory

SYNTAX: **load [<opts>] {<path>}**

FUNCTION: load loads one or more modules specified by <path> into memory.

Unless a full pathlist is specified, <path> is relative to your current execution directory. Consequently, if the module to load is in your execution directory, you only need to enter its name:

load <file>

If <file> is not in your execution directory and if the shell environment variable PATH is defined, load searches each directory specified by PATH until <file> is successfully loaded from a directory. This corresponds to the shell execution search method using the PATH environment variable. By using the -l option, load prints the pathlist of the successfully loaded file.

The names of the modules are added to the module directory. If a module is loaded with the same name as a module already in memory, the module having the highest revision level is kept.

File Security

The OS-9 file security mechanism enforces certain requirements regarding owner and access permissions when loading modules into the module directory.

You must have file access permission to the file to be loaded. If the file is loaded from an execution directory, the execute permission (e) must be set. If the file is loaded from a directory other than the execution directory and the -d option is specified, only the read permission (r) is required.

NOTE: Unless the file has public execute and/or public read permission, only the owner of the file or a super user can load the file. Use the dir -e command to examine a file's owner and access permissions.

You must have module access permission to the file being loaded. This is not to be confused with the file access permission. The module owner and access permissions are stored in the module header; use ident to examine them. To prevent ordinary users from loading super user programs, OS-9 enforces the following restriction: if the module group ID is zero (super group), then the module can be loaded only if the process' group ID and the file's group ID is also zero.

If you are not the owner of a module and not a super user, the public execute and/or read access permissions must be set. The module access permissions are divided into three

groups: the owner, the group, and the public. Only the owner of the module or a super user can set the module access permissions.

OPTIONS:

-?	Displays the options, function, and command syntax of load.
-d	Loads the file from your current data directory, instead of your current execution directory.
-l	Prints the pathlist of the file to be loaded.
-z	Reads the file names from standard input.
-z=<file>	Reads the file names from <file>.

EXAMPLE:

```
$ mdir
   Module Directory at 14:44:35
kernel    init     p32clk    rbf      p32hd
h0        p32fd    d0        d1       ram
r0        dd       mdir

$ load edit

$ mdir
   Module Directory at 14:44:35
kernel    init     p32clk    rbf      p32hd
h0        p32fd    d0        d1       ram
r0        dd       edit      mdir
```

login — Timesharing System Login

SYNTAX: **login [<name>] [,] [<password>]**

FUNCTION: login is used in timesharing systems to provide login security. It is automatically called by the timesharing monitor tsmon, or you can explicitly invoke it after the initial login to change a terminal's user.

login requests a user name and password, which is checked against a validation, or password file. If the information is correct, the user's system priority, user ID, and working directories are set up according to information stored in the file. The initial program specified in the password file is also executed. This initial program is usually the shell. The date, time, and process number are also displayed.

If you cannot supply a correct user name and password after three attempts, the login attempt is aborted.

NOTE: If the shell from which you called login is not needed again, you may discard it using the ex utility to start the login command: ex login.

To log off the system, you must terminate the initial program specified in the password file. For most programs, including shell, you can do this by typing an end-of-file character (escape) as the first character on a line.

If the file SYS/motd exists, a successful login displays the contents of motd on your terminal screen.

The Password File

The password file must be present in the SYS directory being used: /h0/SYS, /d0/SYS, etc. The file contains one or more variable-length text entries; one for each user name. These entries are not shell command lines. Each entry has seven fields. Each field is delimited by a comma. The fields are:

- ¿ **User name.** This field may be up to 32 characters long. It cannot include spaces. The user name may not begin with a number, a period, or an underscore, but these characters may be used elsewhere in the name. If this field is empty, any name matches.

- ¡ **Password.** This field may contain up to 32 characters including spaces. If this field is omitted, no password is required for the specified user.

- ¬ **Group.User ID number.** This field allows 0 to 65535 groups and 0 to 65535 users. 0.n is the super user. The file security system uses this number as the system-wide user ID to identify all processes initiated by the user. The system manager should assign a unique ID to each potential user.

- Ð **Initial process priority:** The initial process priority can be from 1 to 65535.

- *f* **Initial execution directory pathlist**. The initial execution directory is usually /d0/CMDS. Specifying a period (.) for this field defaults the initial execution directory to the CMDS file located in the current directory, usually /h0 or /d0.

- Ý **Initial data directory pathlist**. This is the specific user directory. Specifying a period (.) for this field defaults to the current directory.

- ý **Initial Program.** The name and parameters of the program to initially execute. This is usually shell.

Sample Password File:

```
superuser,secret,0.0,255,.,.,shell -p="@howdy"
brian,open sesame,3.7,128,../d1/STEVE,shell
sara,jane,3.10,100,/d0/BUSINESS,/d1/LETTERS,wordprocessor
robert,,4.0,128,./d1/ROBERT,Basic
mean_joe,midori,12.97,100,Joe,Joe,shell
```

Using password file entries, login sets the following shell environment variables. Programs can examine these environment variables to determine various characteristics of the user's environment:

Name	Description

HOME	Initial data directory pathlist
SHELL	Name of the initial program executed
USER	User name
PATH	Login process' initial execution directory. If a period (.) is specified, PATH is not set.

NOTE: Environment variables are case sensitive.

To show how login uses the password file to set up environment variables, examine the previous sample password file. Assume login's data and execution directories are /h0 and /h0/CMDS, respectively, logging in as mean_joe executes a shell with the data directory of /h0/Joe and the execution directory of /h0/CMDS/Joe. The environment variables passed to the shell are set as follows:

```
HOME=/h0/Joe
SHELL=shell
USER=mean_joe
PATH=/H0/Cmds
```

OPTION: -? Displays the function and command syntax of login.

logout — Timesharing System Logout

SYNTAX: logout

FUNCTION: logout terminates the current shell. If the shell to terminate is the login shell, logout executes the .logout procedure file before terminating the shell.

To terminate the current shell, type logout and a carriage return. This terminates the current shell in the same manner as an end-of-file character, with one exception. If the shell to be terminated is the login shell, logout executes the procedure file .logout. The login shell is the initial shell created by the login utility when you log on the system. In order for logout to execute the .logout file, .logout must be located in the directory specified by the HOME environment variable.

EXAMPLE:

```
3.lac list .logout
procs
wait
date
echo "see you later. . ."
3.lac logout
2.lac logout
1.lac logout
 Id PId Grp.Usr Prior MemSiz Sig S   CPU Time   Age Module & I/O
  2   1   0.0   128    0.25k   0 w       0.01   ??? sysgo <>>>term
  3   2   0.0   128    4.75k   0 w       4.11 01:13 shell <>>>term
  4   3   0.0   128    8.50k   0 *       0.08 00:00 procs <>>term
  5   0   0.0   128    4.00k   0 s       0.02 01:12 tsmon <>>>t1
July 7, 1989 11:59 pm
see you later . . .
```

makdir — Create a Directory File

SYNTAX: **makdir [<opts>] {<path>}**

FUNCTION: makdir creates a new directory file specified by the given pathlist.

To create a new directory, type makdir and the pathlist specifying the new directory. You must have write permission for the new directory's parent directory. The new directory is initialized and does not initially contain files except for the pointers to itself (.) and its parent directory (.). All access permissions are enabled except single use, or non-sharable.

It is an OS-9 convention to capitalize directory names to distinguish them from lower case file names. This is not required; it is just a convention.

OPTIONS:

-?	Displays the options, function, and command syntax of makdir.
-x	Creates the directory in the execution directory.
-z	Reads the directory names from standard input.
-z=<file>	Reads the directory names from <file>.

EXAMPLES:

```
$ makdir /d1/STEVE/PROJECT
$ makdir DATAFILES
$ makdir ../SAVEFILES
$ makdir RED GREEN BLUE ../PURPLE
```

make — Maintain, Update, and Regenerate Groups of Programs

SYNTAX: **make [<opts>] [<target>] {[<target>]} [<macros>]**

FUNCTION: make determines whether a file needs to be updated. It examines the dates of the target file and the files used to create the target file. If make determines that the file must be updated, it executes specified commands to re-create the file. make has several built-in assumptions specifically designed for compiling high-level language programs; however, you may use make to maintain any files dependent on updated files.

make executes commands from a special type of procedure file called a ***makefile***. The makefile describes the dependent relationships between files used to create the <target> file(s). A makefile may describe the commands to create many files. If make is invoked without a target file on the command line, make attempts to make the first target file described in the makefile. If one or more target file's are entered on the command line, make reads and processes the entire makefile and only attempts to make the appropriate file(s).

A makefile contains three types of entries:

¿ ***Dependency Entry:*** This specifies the relationship of a target file and the file(s) used to build the target file. The entry has the following syntax:

<target>:[[<file>],<file>]

The list of files following the target file is known as the ***dependency list***.

¡ ***Command Entry:*** This specifies the command that you must execute to update a particular target file, if the target file needs to be updated. make updates a target file only if it depends on files newer than itself.

If no instructions for update are provided, make attempts to create a command entry to perform the operation. make recognizes a command entry by a line beginning with one or more spaces or tabs. Any legal OS-9 command line is acceptable. You can list more than one command entry for any dependency. Each command is forked separately unless continued from the previous command with a backslash (\). Do not intersperse comments with commands. For example:

```
<target>:[[<file>],<file>]
    <OS-9 command line>
    <OS-9 command line>
```

- ***Comment Entry:*** This consists of any line beginning with an asterisk (*). All characters following a pound sign (#) are also ignored as comments with one exception: a digit following a pound sign is considered part of a command entry. All blank lines are ignored. For example:

  <target>:[[<file>],<file>]

  *** the following command will be executed if the dependent**
  *** files are newer than the target file**
  <OS-9 command line> # this is also a comment line

You may continue entries on the next line by placing a space followed by a backslash (\) at the end of each line to continue. If a command line is continued, a space or tab must be the first character in the continued line. With non-command lines, leading spaces and tabs are ignored on continuation lines. Entries longer than 256 characters must be continued on the next line. For example:

**FILES = aaa.r bbb.r ccc.r ddd.r eee.r fff.r ggg.r **
hhh.r iii.r jjj.r

make starts by reading the entire makefile and setting up a table of dependencies exactly as listed in the makefile. When make encounters a name on the left side of a colon, it first checks to see if it has encountered the name before. If it has, make connects the lists and continues.

After reading the makefile, make determines the target file(s). The target file is the main file to be made on the list. It then makes a second pass through the dependency table. During the second pass, make looks for object files with no relocatable files in their dependency lists and for relocatable files with no source files in their dependency lists. This facilitates program compilation. If make needs to find any source files or relocatable files to complete the dependency lists, it looks for them in the specified data directory, unless a macro is specified.

make does a third pass through the list to get the file dates and compare them. When make finds a file in a dependency list that is newer than its target file, it executes the specified command(s). If no command entry is specified, a command is generated based on the assumptions given in the next section. Because OS-9 only stores the time down to the closest minute, make remakes a file if its date matches one of its dependents.

When a command is executed, it is echoed to standard output unless the -s, or silent, option is used or the command line starts with an "at" sign (@). When the -n option is used, the command is echoed to standard output but is not actually executed.

If your system runs out of memory while executing a command, you can redirect the output of make into a procedure file and execute the procedure file.

make normally stops if an error code is returned when a command line is executed. Errors are ignored if the -i option is used or a command line begins with a hyphen.

Sometimes, it is helpful to see the file dependencies and the dates associated with each of the files in the list. The -d option turns on the make debugger and gives a complete listing of the macro definitions, a listing of the files as it checks the dependency list, and all the file modification dates. If it cannot find a file to examine its date, it assumes a date of -1/00/00 00:00, indicating the necessity to update the file.

To update the date on a file without remaking it, use the -t option. make merely opens the file for update and then closes it, thus making the date current.

If you are quite explicit about your makefile dependencies and do not want make to assume anything, use the -b option to turn off the built-in rules governing implicit file dependencies.

Implicit Rules, Definitions, and Assumptions

Any time a command line is generated, make assumes the target file is a program to compile. Therefore, if the target file is not a program to compile, make sure the command entries are included for each dependency list. make uses the following definitions and rules when forced to create a command line.

Object Files: Files with no suffixes. An object file is made from a relocatable file and is linked when it needs to be made.

Relocatable Files: Files appended by the suffix .r. Relocatable files are made from source files and are assembled or compiled if they need to be made.

Source Files: Files having one of the following suffixes: .a, .c, .f, or .p.

Default Compiler: cc

Default Assembler: r68

Default Linker: cc

Default Directory for All Files: current data directory (.)

NOTE: Only use the default linker with programs that use Cstart.

Macro Recognition

make recognizes a macro by the dollar sign ($) character in front of the name. If a macro name is longer than a single character, the entire name must be surrounded by parentheses. For example, $R refers to the macro R, $(PFLAGS) refers to the macro PFLAGS, $(B) and $B refer to the macro B, and $BR is interpreted as the value for the macro B followed by the character R.

Macros may be placed in the makefile for convenience or on the command line for flexibility. Everywhere the macro name appears, the expansion is substituted for it. Macros are allowed in the form of <macro name> = <expansion>.

NOTE: If a macro is defined in your makefile and then redefined on the command line, the command line definition overrides the definition in the makefile. This feature is useful for compiling with special options.

In order for make to be more flexible, you can define special macros in the makefile. make uses these macros when assumptions must be made in generating command lines or searching for unspecified files. For example, if no source file is specified for program.r, make searches the specified directory, SDIR, or ".", for program.a (or .c, .p, .f).

make recognizes the following special macros:

Macro	Definition
ODIR=<path>	make searches the directory specified by <path> for all files that have no suffix or relative pathlist. If ODIR is not defined in the makefile, make searches the current directory by default.
SDIR=<path>	make searches the directory specified by <path> for all source files not specified by a full pathlist. If SDIR is not defined in the makefile, make searches the current directory by default.
RDIR=<path>	make searches the directory specified by <path> for all relocatable files not specified by a full pathlist. If RDIR is not defined, make searches the current directory by default.
CFLAGS=<opts>	These compiler options are used in any necessary compiler command lines.
RFLAGS=<opts>	These assembler options are used in any necessary assembler command lines.
LFLAGS=<opts>	These linker options are used in any necessary linker command lines.
CC=<comp>	make uses this compiler when generating command lines. The default compiler is cc.
RC=<asm>	make uses this assembler when generating command lines. The default assembler is r68.
LC=<link>	make uses this linker when generating command lines. The default linker is cc.

Some reserved macros are expanded when a command line associated with a particular file dependency is forked. These macros may only be used on a command line. They are useful when you need to be explicit about a command line but have a target program with several dependencies.

In practice, these reserved macros are wildcards with the following meanings:

Macro	Definition
$@	Expands to the name of the file made by the command.
$*	Expands to the prefix of the file made.
$?	Expands to the list of files found to be newer than the target on a given dependency line.

Make Generated Command Lines

make is capable of generating three types of command lines:

¿ Compiler Command Lines: These are generated if a source file with a suffix of .c, .f, or .p needs to be recompiled. The compiler command line generated by make has the following syntax:

$(CC) $(CFLAGS) -r=$(RDIR) $(SDIR)/<file>[.c, .f, or .p]

¡ Assembler Command Lines: These are generated if an assembly language source file needs to be re-assembled. The assembler command line generated by make has the following syntax:

$(RC) $(RFLAGS) $(SDIR)/<file>.a -o=$(RDIR)/<file>.r

¬ Linker Command Lines: These are generated if an object file needs to be relinked in order to re-make the program module. The linker command line generated by make has the following syntax:

$(LC) $(LFLAGS) $(RDIR)/<file>.r -f=$(ODIR)/<file>

WARNING: When make is generating a command line for the linker, it looks at its list and uses the first relocatable file it finds, but only the first one. For example:

prog: x.r y.r z.r

would generate

cc x.r, not **cc x.r y.r z.r** or **cc prog.r**

OPTIONS:

-?	Displays the options, function, and command syntax of make.
-b	Does not use built-in rules.
-bo	Does not use built-in rules for object files.
-d	Debug Mode. Prints the dates of the files in the makefile.
-dd	Double debug mode, very verbose.
-f-	Reads the makefile from standard input.
-f=<path>	Specifies <path> as the makefile. If <path> is specified as a hyphen (-), make commands are read from standard input.
-i	Ignores errors.
-n	Does not execute commands, but does display them.
-s	Silent Mode. Executes commands without echo.
-t	Updates the dates without executing commands.
-u	Does the make regardless of the dates on files.
-x	Uses the cross-compiler/assembler.
-z	Reads a list of make targets from standard input.
-z=<path>	Reads a list of make targets from <path>.

Options may be included on the command line when running make, or they may be included in the makefile for convenience.

CAVEAT: The make language is highly specific. Therefore, use caution when using dummy files with names like print.

CAVEAT: make is always case-dependent with respect to directory names and file names.

Unless a file is specifically an object file or the -b option is used to turn off the implicit rules, use a suffix for your dummy files. For example, use print.file and xxx.h for your header files.

mdir — Display Module Directory

SYNTAX: **mdir [<opts>] [<modname>]**

FUNCTION: mdir displays the present module names in the system module directory. The system module directory contains all modules currently resident in memory. By specifying individual module names, only specified modules are displayed if resident in memory.

If you use the -e option, an extended listing of the physical address, size, owner, revision level, user count, and the type of each module is displayed.

The module type is listed using the following mnemonics:

Mnemonic	Type of Module
Prog	Program Module
Subr	Subroutine Module
Mult	Multi Module
Data	Data Module
Trap	Trap Handler Module
Sys	System Module
FMan	File Manager
Driv	Device Driver Module
Desc	Device Descriptor Module

NOTE: User-defined modules not corresponding with this list are displayed by their number.

By using the -a option, the language of each module is displayed instead of the type in an extended listing. The language field uses the following mnemonics:

Mnemonic	Module Language
Obj	68000 Machine Code
Bas	Basic09 I Code
Pasc	Pascal I Code
C	C I Code
Cobl	Cobol I Code
Fort	Fortran I Code

NOTE: If the language field is inappropriate for the module, a blank field is displayed. For example, d0, t1, or init.

WARNING: Not all modules listed by mdir are executable as processes; always check the module type code to make sure it is executable before executing an unfamiliar module.

OPTIONS:

-?	Displays the options, function, and command syntax of mdir.
-a	Displays the language field instead of the type field in an extended listing.
-e	Displays the extended module directory.
-t=<type>	Displays only the modules of the specified type.
-u	Displays an unformatted listing used for piping the output etc.

EXAMPLES: To save space, the following examples are fairly incomplete. Module directories are generally much larger.

```
$ mdir
   Module Directory at 15:32:38

kernel    syscache   ssm       init      tk147
rtclock   rbf

$ mdir -e
 Addr     Size     Owner    Perm Type Revs  Ed #  Lnk  Module name
-------- -------- ----------- ---- ---- ---- ----- ----- --------------
00006f00    27562     0.0     0555 Sys  a000    83     2 kernel
0000daaa      368     0.0     0555 Sys  a000    10     1 syscache
0000dc1a     1682     0.0     0555 Sys  a000    29     1 ssm
0000e2ac      622     0.0     0555 Sys  8000    20     0 init
0000e51a      322     0.0     0555 Sys  a000     7     1 tk147
0000e65c      494     0.0     0555 Subr a000     8     0 rtclock
0000e84a     8952     0.0     0555 Fman e000    79    26 rbf

$ mdir -ea
 Addr     Size     Owner    Perm Lang Revs  Ed #  Lnk  Module name
-------- -------- ----------- ---- ---- ---- ----- ----- --------------
00006f00    27562     0.0     0555 Obj  a000    83     2 kernel
0000daaa      368     0.0     0555 Obj  a000    10     1 syscache
0000dc1a     1682     0.0     0555 Obj  a000    29     1 ssm
0000e2ac      622     0.0     0555      8000    20     0 init
0000e51a      322     0.0     0555 Obj  a000     7     1 tk147
0000e65c      494     0.0     0555 Obj  a000     8     0 rtclock
0000e84a     8952     0.0     0555 Obj  e000    79    26 rbf
```

merge — Copy and Combine Files to Standard Output

SYNTAX: **merge [<opts>] {<path>}**

FUNCTION: merge copies multiple input files specified by <path> to standard output. merge is commonly used to combine several files into a single output file.

Data is copied in the order the pathlists are specified on the command line. merge does no output line editing such as automatic line feed. The standard output is generally redirected to a file or device.

OPTIONS:

-?	Displays the options, function, and command syntax of merge.
-b=<num>	Allocates a <num>k buffer size for use by merge. The default memory size is 4K.
-x	Searches the current execution directory for files to be merged.
-z	Reads the file names from standard input.
-z=<file>	Reads the file names from <file>.

EXAMPLES:

```
$ merge compile.list asm.list >/p

$ merge file1 file2 file3 file4 >combined.file -b=32k

$ merge -x load link copy >Utils1

$ merge -z=/h0/PROGS/file1 >merged_files
```

mfree — Display Free System RAM

SYNTAX: **mfree [<opts>]**

FUNCTION: mfree displays a list of areas in memory not presently in use and available for assignment. The address and size of each free memory block are displayed.

OPTIONS:

-? Displays the option, function, and command syntax of mfree.

-e Displays an extended free memory list.

EXAMPLES:

```
$ mfree
Current total free RAM: 1392.00 K-bytes

mfree -e
Minimum allocation size:      4.00 K-bytes
Number of memory segments:  25
Total RAM at startup:      4095.00 K-bytes
Current total free RAM:    1392.00 K-bytes

Free memory map:

  Segment Address       Size of Segment
 ----------------  --------------------------
    $55000         $7000        28.00 K-bytes
    $6A000         $B000        44.00 K-bytes
    $80000         $8A000      552.00 K-bytes
    $10E000        $1A000      104.00 K-bytes
    $12F000        $1E000      120.00 K-bytes
    $151000        $60000      384.00 K-bytes
    $1B5000        $2000         8.00 K-bytes
    $1B8000        $E000        56.00 K-bytes
    $1DE000        $1000         4.00 K-bytes
    $208000        $4000        16.00 K-bytes
    $21C000        $5000        20.00 K-bytes
    $245000        $1000         4.00 K-bytes
    $249000        $1000         4.00 K-bytes
```

moded — Edit OS-9 Modules

SYNTAX: **moded [<opts>] [<path>]**

FUNCTION: moded is used to edit individual fields of certain types of OS-9 modules. Currently, you can use moded to change the Init module and any OS-9 device descriptor module. moded can edit modules which exist in their own files and modules which exist among other modules in a single file such as a bootstrap file. moded updates the module's CRC and header parity if changes are made.

Regardless of how you invoke moded, you always enter the editor's command mode. This is designated by the moded: prompt.

If no parameters are specified on the moded command line, no current module is loaded in memory.

If a file is specified on the command line, it is assumed to contain a module of the same name. This module is loaded into the editor's buffer and becomes the ***current*** module.

If the -f option is used, the specified file is loaded into the editor's memory. If a module of the same name exists in the file, it becomes the current module. If no such module exists, there is no current module.

If the -f option is used and a module name is specified on the command line, the specified module becomes the current module.

The following commands may be executed from command mode:

Command	Description
e(dit)	Edits the current module.
f(ile)	Opens a file of modules.
l(ist)	Lists the contents of the current module.
m(odule)	Finds a module in a file.
w(rite)	Updates the module CRC and writes to the file.
q(uit)	Returns to the shell.
$	Calls the OS-9 shell.
?	Prints this help message.

Once moded is invoked, it attempts to read the moded.fields file. This file contains module field information for each type of module to edit. Without this file, moded cannot function.

moded searches for moded.fields in the following directories in this order:

- Search device /dd first.
- Search the default system device, as specified in the Init module (M$SysDev). If the Init module cannot be linked to, the SYS directory is searched for on the current device.

If this file cannot be found, an error is returned.

Selecting the Current Module

If you do not specify a module or file on the command line, you may open a module or file from command mode using the e or f commands, respectively. The e command prompts for a file name and a module name if different from the file name. This module then becomes the current module.

The f command prompts for the name of a file containing one or more modules. If a module in the file has the same name as the file, it becomes the current module by default. Use the m command to change the current module.

Edit Mode

To edit the current module, use the e command. If there is no current module, the editor prompts for the module name to edit. The editor prints the name of a field, its current value, and prompts for a new value. At this point, you can enter any of the following edit commands:

Command	Description
<expr>	A new value for the field.
-	Re-displays the last field.
.	Leaves the edit mode.
?	Prints the edit mode commands.
??	Prints a description of the current field.
<cr>	Leaves the current value unchanged.

If the definition of any field is unfamiliar, use the ?? command for a short description of the current field.

Once you have made all necessary changes to the module, exit edit mode by reaching the end of the module or by typing a period. At this point, the changes made to the module exist only in memory. To write the changes to the actual file, use the w command. This also updates the module header parity and CRC.

Listing Module Fields

To examine the field values of the current module, use the l command. This displays a formatted list of the field names and their values.

The Moded.fields File

The moded.fields file consists of descriptions of specific types of modules. Each module description consists of three parts: the module type, the field descriptor, and the description lines. Comments may be interspersed throughout the file by preceding the comment line with an asterisk. For example:

```
* this is a comment line
* it may appear anywhere in the moded.fields file
```

¿ ***The Module Type:*** This is a single line consisting of the module type as specified in M$Type in the module header and the device type as specified in PD_DTP in the device descriptor. Both values are specified as decimals and are separated from each other by a comma. The module type line is the only line which begins with a pound sign (#). The following example line describes an RBF device descriptor module:

```
#15,1
```

Two module type values are accepted:

Value	Description
12	System Module (Init module only)
15	Device Descriptor Module

The device type value is only used when a device descriptor module is being described. The following device type values are accepted:

Value	Description
0	SCF
1	RBF
2	PIPE
3	SBF
4	NET
6	UCM
11	GFM

¡ ***The Field Descriptor:*** This consists of two lines. The first is a textual description of the module field; the baud rate, parity. and descriptor name. moded uses this description as a prompt to change this field's value.

The second line has the following format:

<type>,<offset>,<base>,<value>[,<name>]

<type> specifies the field size in bytes. This is a decimal value. The following values are accepted:

1	byte
2	word
3	3 byte value
4	long word
5	long word offset to a string
6	word offset to a string

<offset> specifies the offset of the field from the beginning of the module. This is a hexadecimal value. **NOTE:** For device-specific fields (see <name> below), this offset is the offset of the field within the DevCon section of the descriptor (and not the module start).

<base> specifies the numeric base in which the field value is displayed in moded. The following bases are supported:

0	ASCII
8	Octal
10	Decimal
16	Hexadecimal

<value> specifies the default value of the field. This is currently unused; set it to zero.

<name> specifies the driver name for this and each field description that follows until a new <name> is specified or a module type line is encountered. This field is optional. For example, <name> allows descriptors with DevCon sections specific to certain drivers to be edited.

The following lines describe a "descriptor name" field:

descriptor name
5,c,0,0

The field consists of a long-word offset to a string. It is offset 12 bytes from the beginning of the module. The display base is in ASCII.

- ***Description Lines:*** After the Field Descriptor lines, you can use any number of lines to describe the field. This description is displayed when the edit mode command, ??, is used. Each description line must begin with an exclamation point (!) to differentiate it from a Field Descriptor. These lines are optional, but they are useful when editing uncertain module fields. The following lines might be used to describe the example used for the Field Descriptor:

```
! This field contains the name that the descriptor
! will be known by when in memory.
```

Example Module Description in Moded.fields:

The following example shows how you could set up a module description:

```
*************************************************************
*the following section describes an RBF device descriptor *
*************************************************************
#15,1
descriptor name
5,c,0,0
! This field contains the name that the descriptor will
! be known by when in memory.
port address
4,30,16,0
! This is the absolute physical address of the hardware
! controller.
irq vector
1,34,10,0
! This is the irq vector that the device will assert.
! Auto-vectored interrupt devices will use vectors 25-31.
! Vectored interrupt devices will use vectors 64-255.
```

The Provided Moded.fields File:

The provided moded.fields file comes with module descriptions for standard RBF, SBF, SCF, PIPE, NETWORK, UCM, and GFM module descriptors. It also includes a description for the Init module.

OPTIONS:

-?	Displays the help message.
-d=<path>	Use <path> for the field descriptions (moded.fields).
-e=<path>	Use <path> for the error message file.
-f=<path>	Specifies a file consisting of one or more modules to be loaded into the moded buffer.

os9gen — Build and Link a Bootstrap File

SYNTAX: **os9gen [<opts>] <devname> {<path>}**

FUNCTION: os9gen creates and links the OS9Boot file required on any disk from which OS-9 is to be bootstrapped. You can use os9gen to make a copy of an existing boot file, add modules to an existing boot file, or create an entirely new boot file for a different system. These are just a few examples.

To use the os9gen utility, type os9gen and the name of the device on which the OS9Boot file is to be installed. os9gen creates a working file called TempBoot on the device specified. Each file specified on the command line is opened and copied to the TempBoot file.

NOTE: Only super users (0.n) may use this utility. os9gen can also only be used on format-enabled devices.

After all input files are copied to TempBoot, any existing OS9Boot file on the target device is renamed OldBoot. If an OldBoot file is already present, it is deleted before OS9Boot is renamed.

TempBoot is then renamed OS9Boot. Its starting address and size are linked in the disk's Identification Sector (LSN 0) for use by the OS-9 bootstrap firmware.

If your boot file is non-contiguous or larger than 64K, use the -e option. **NOTE:** Your bootstrap ROMs must support this feature. If they do not, you should not use this option.

If the -z option is used, os9gen first uses the files specified on the command line and then the file names from its standard input, or from the specified pathlist, one pathlist per line. If the names are entered manually, no prompts are given and the end-of-file key (usually <escape>) or a blank line is entered after the line containing the last pathlist.

To determine what modules are necessary for your boot file, use the ident utility with the OS9Boot file that came with your system.

The -q option updates information in the disk's Identification Sector by directing it to point to a file already contained in the root directory of the specified device.

The -q option is useful when restoring the OldBoot file as the valid boot on the disk. os9gen renames the specified file to be OS9Boot and saves the current boot as described previously.

The -r option removes the pointer to the boot file but does not delete the file. This is useful if you delete the bootfile from your disk (using the del command). Deleting the bootfile from the file structure *does not* remove the bootfile pointers from the disk's Identification Sector. It can also be used to make a disk non-bootable without deleting the actual bootfile.

OPTIONS:

-?	Displays the options, function, and command syntax of os9gen.
-b=<num>	Assigns <num>k of memory for os9gen. Default memory size is 4K.
-e	Extended Boot. Allows you to use large (greater than 64K) and/or non-contiguous files. **NOTE:** Bootstrap ROMs must support this feature.
-q=<file>	Quick Boot. Sets sector zero pointing to <file>.
-r	Removes the pointer to the boot file. This file is not deleted.
-x	Searches the execution directory for pathlists.
-z	Reads the file names from standard input.
-z=<file>	Reads the file names from <file>.

EXAMPLES: This command manually installs a boot file on device /d1 which is an exact copy of the OS9Boot file on device /d0.

```
$ os9gen /d1 /d0/os9boot
```

The following three methods manually install a boot file on device /d1. The boot file on /d1 is a copy of the OS9Boot file on device /d0 with the addition of modules stored in the files /d0/tape.driver and /d2/video.driver:

Method 1:

```
$ os9gen /d1 /d0/os9boot /d0/tape.driver /d2/video.driver
```

Method 2:

```
$ os9gen /d1 /d0/os9boot -z
/d0/tape.driver
/d2/video.driver
[ESCAPE]
```

Method 3:

```
$ os9gen /d1 -z
/d0/os9boot
/d0/tape.driver
/d2/video.driver
[ESCAPE]
```

You can automatically install a boot file by building a ***bootlist*** file and using the -z option to either redirect os9gen standard input or use the specified file as input:

$ build /d0/bootlist	*Create file* bootlist
? /d0/os9boot	*Enter first file name*
? /d0/tape.driver	*Enter second file name*
? /d2/video.driver	*Enter third file name*
? * V1.2 of video driver	*Comment line*
? [RETURN]	*Terminate* build
$ os9gen /d1 -z </d0/bootlist	*Redirects standard input*
$ os9gen /d1 -z=/d0/bootlist	*Reads input from pathlist*

NOTE: os9gen treats any input line preceded by an asterisk (*) as a comment.

The following command makes the OldBoot file the current boot and saves the current OS9BOOT file as OldBoot:

```
$ os9gen /d1 -q=oldboot
```

pd — Print the Working Directory

SYNTAX: **pd [<opts>]**

FUNCTION: pd displays a pathlist showing the path from the root directory to your current data directory. Programs can use pd to discover the actual physical location of files or by users to find their whereabouts in the file system. pd -x displays the pathlist from the root directory to the current execution directory.

OPTIONS:

-? Displays the option, function, and command syntax of pd.

-x Displays the path to the current execution directory.

EXAMPLES:

```
$ chd /D1/STEVE/TEXTFILES/MANUALS
$ pd
/d1/STEVE/TEXTFILES/MANUALS

$ chd ..
$ pd
/d1/STEVE/TEXTFILES

$ chd ..
$ pd
/d1/STEVE

$ pd -x
/d0/CMDS
```

pr Print Files

SYNTAX: **pr [<opts>] {<path>}**

FUNCTION: pr produces a formatted listing of one or more files to the standard output.

To use the pr utility, type pr and the pathlist(s) of the files to list. The listing is separated into pages. Each page has the page number, the name of the listing, and the date and time printed at the top.

pr can produce multi-column output. When printing multiple output columns with the -m option, if an output line exceeds the column width, the output line is truncated. pr can also print files simultaneously, one per column.

If no files are specified on the command line and the -z option is used, standard input is assumed to be a list of file names, one file name per input line, to print out. If no files are specified on the command line and the -z option is not used, standard input is displayed on standard output.

Files and options may be intermixed.

A typical page of output consists of 66 lines of output. Consequently, pr uses the following default parameters: 61 lines of output with 5 blank lines as a trailer. The 61 lines of output contain one line for the title, 5 blank lines for a header, and 55 lines of text. The trailer can be reduced or eliminated by expanding the number of lines per page.

OPTIONS: An equal sign (=) in an option specification is optional.

-?	Displays the options, function, and command syntax of pr.
-c=<char>	Uses <char> as the specified column separator. A <space> is the default column separator.
-d	Specifies the actual page depth.
-f	Pads the page using a series of \n (new line), instead of a \f (form feed).
-h=<num>	Sets the number of blank lines after title line. The default is 5.
-k=<num>	Sets the <num> columns that the output file will be listed in for multi-column output.
-l=<num>	Sets the left margin to <num>. The default is 0.
-m	Prints files simultaneously, one file per column. If three files are given on the command line, each file is printed in its own column on the page.
-n=<num>	Specifies the line numbering increment: <num>. The default is 1.

-o	Truncates lines longer than the right margin. By default, long lines are wrapped around to the next line.
-p=<num>	Specifies the number of lines per page: <num>. The default is 61.
-r=<num>	Sets the right margin to <num>. The default is 79.
-t	Does not print title.
-u=<title>	Uses specified title instead of file name. <title> may not be longer than 48 characters.
-x=<num>	Sets the starting page number to <num>. The default is 1.
-z	Reads the file names from standard input.
-z=<file>	Reads the file names from <file>.

EXAMPLES: The following example prints file1 using the default values of 55 lines of text per page, one line for the title, and 5 lines each for the header and trailer:

```
$ pr file1 >/p1
```

The following example prints file1 with no title. This uses 56 lines of text per page:

```
$ pr file1 -t >/p1
```

The following example prints file1 using 90 lines per page. Pagination begins with page 10:

```
$ pr file1 -x=10 p=90 >/p1
```

To display a numbered, unformatted listing of the data directory, type:

```
$ dir -u ! pr -n
```

printenv — Print Environment Variables

SYNTAX: **printenv**

FUNCTION: printenv prints any defined environment variables to standard output.

EXAMPLE: **$ printenv**

```
NAME=andy
TERM=abm85
LIST=/p1
As_long_as_you_want=long_value
```

SEE ALSO: setenv and unsetenv utility descriptions and the discussion of the shell environment in the chapter on the shell

procs — Display Processes

SYNTAX: **procs [<opts>]**

FUNCTION: procs displays a list of processes running on the system owned by the user invoking the routine. Processes can switch states rapidly, usually many times per second. Consequently, the display is a snapshot taken at the instant the command is executed and shows only those processes running at that exact moment.

procs with no options displays ten pieces of information for each process:

Id	Process ID
PId	Parent process ID
Grp.usr	Owner of the process (group and user)
Prior	Initial priority of the process
MemSiz	Amount of memory the process is using
Sig	Number of any pending signals for the process
S	Process status: w Waiting s Sleeping a Active * Currently executing
CPU Time	Amount of CPU time the process has used
Age	Elapsed time since the process started
Module & I/O	Process name and standard I/O paths: < Standard input > Standard output >> Standard error output If several of the paths point to the same pathlist, the identifiers for the paths are merged.

procs -a displays nine pieces of information: the process ID, the parent process ID, the process name, and standard I/O paths and six new pieces of information:

Aging	Age of the process based on the initial priority and how long it has waited for processing
F$calls	Number of service request calls made
I$calls	Number of I/O requests made
Last	Last system call made
Read	Number of bytes read
Written	Number of bytes written

The -b option displays both sets of information. The -e option displays information for all processes in the system. Detailed explanation of all information displayed by procs is available in the ***OS-9 Technical Manual***.

OPTIONS:

-? Displays the options, function, and command syntax of procs.

-a Displays alternate information.

-b Displays regular and alternate procs information.

-e Displays all processes of all users.

EXAMPLES:

```
$ procs
Id PId Grp.Usr Prior MemSiz Sig S   CPU Time   Age Module & I/O
 2   1   0.0   128   0.25k   0 w      0.01   ??? sysgo <>>>term
 3   2   0.0   128   4.75k   0 w      4.11 01:13 shell <>>>term
 4   3   0.0     5   4.00k   0 a   12:42.06 00:14 xhog <>>>term
 5   3   0.0   128   8.50k   0 *      0.08 00:00 procs <>>term
 6   0   0.0   128   4.00k   0 s      0.02 01:12 tsmon <>>>t1
 7   0   0.0   128   4.00k   0 s      0.01 01:12 tsmon <>>>t2

$ procs -a
Id PId  Aging  F$calls I$calls Last     Read Written Module & I/O
 2   1   129        5       1 Wait        0       0 sysgo <>>>term
 3   2   132      116     127 Wait      282     129 shell <>>>term
 4   3    11        1       0 TLink       0       0 xhog  <>>>term
 5   3   128        7       4 GPrDsc      0       0 procs <>>>term
 6   0   130        2       7 ReadLn      0       0 tsmon <>>>t1
 7   0   129        2       7 ReadLn      0       0 tsmon <>>>t2
```

profile — Read Commands from File and Return

SYNTAX: **profile <path>**

FUNCTION: profile causes the current shell to read its input from the named file and then return to its original input source which is usually the keyboard.

The file specified in <path> may contain any utility or shell commands, including those to set or unset environment variables or to change directories. These changes remain in effect after the command executes. This is in contrast to calling a normal procedure file by name only, which would then be executed by a child shell. This would not affect the environment of the calling shell.

You can nest profile commands. That is, the file itself may contain a profile command for another file. When the latter profile command is completed, the first one will resume.

A particularly useful application for profile files is within the .login and .logout files of a system's users. For example, if each user includes the following line in their .login file, system-wide commands (common environments, news bulletins, etc.) can be included in the file /dd/SYS/login_sys:

```
profile /dd/SYS/login_sys
```

A similar technique can be used for .logout files.

qsort — In-Memory Quick Sort

SYNTAX: **qsort [<opts>] {<path>}**

FUNCTION: qsort is a quick sort algorithm that sorts any number of lines up to the maximum capacity of memory.

To use qsort, type qsort and the pathlist(s) of the file(s) to sort. qsort sorts the file(s) by a user-specified field or field one by default. The field separation character defaults to a space if no separation character is specified. If no file names are given on the command line, standard input is assumed.

CAVEAT: Multiple separation characters in a row are counted as a single field separator. For example, if a comma is specified as the field separation character, three commas in a row (,,,) signify only one field separator. If the intent is to create two null fields, a space must be inserted between each comma (, , ,).

OPTIONS:

-?	Displays the options, function, and command syntax of qsort.
-c=<char>	Specifies the field separation character. If an asterisk (*), question mark (?), or comma (,) are used as field separation characters, the option and the character must be enclosed by quotation marks.
-f=<num>	Specifies the sort field. **NOTE:** Only one -f field is allowed on a command line.
-z	Reads the file names from standard input.
-z=<file>	Reads the file names from <file>.

EXAMPLES:

$ qsort file1 file2 file3	Sorts files and displays.
$ dir -ue ! qsort -f=7	Sorts extended directory listing by entry name, field 7.
$ qsort file -f=2 "-c=*"	Sorts file by field 2 using an asterisk (*) as the field separation character.
$ qsort file -f=2 "-c=,"	Sorts file by field 2 using a comma (,) as the field separation character.
$ qsort -z	Reads file names from standard input.

rename — Change File Name

SYNTAX: **rename [<opts>] <path> <new name>**

FUNCTION: rename assigns a new name to the mass storage file specified in the pathlist.

To rename a file, type rename, followed by the name of the file to rename, followed by the new name. You must have write permission for the file to change its name. You cannot use the names "." or ".." for <path>.

OPTIONS:

-? Displays the option, function, and command syntax of rename.

-x Indicates that <path> starts at the current execution directory. You must have execute permission for the specified file.

EXAMPLES:

```
$ dir
  Directory of .  16:22:53
blue          myfile
$ rename blue purple
$ dir
  Directory of .  16:23:22
myfile        purple

$ rename /h0/HARRY/test1 test2

$ rename -x screenclear clearscreen
```

romsplit — Split File

SYNTAX: **romsplit {<opts>} {<path>}**

FUNCTION: romsplit splits the input file specified by <path> into two or four files.

romsplit converts a ROM object image into an 8-bit wide file. This is useful when a PROM programmer cannot burn more than one PROM at a time and the system has the ROMs addressed as 16-bit or 32-bit wide memory.

If the -q option is not specified, romsplit copies the even bytes of data to a new file with the same name with a .0 extension. The odd bytes are copied to a new file with the same name with a .1 extension.

If the -q option is specified, the following copying scheme is used:

Byte Number	Destination File
0, 4, 8, 12 etc.	<path>.0
1, 5, 9, 13 etc.	<path>.1
2, 6, 10, 14 etc.	<path>.2
3, 7, 11, 15 etc.	<path>.3

OPTIONS:

-? Displays the options, function, and command syntax of romsplit.

-q Splits the input file into four files.

-x Reads the input file from execution directory.

save — Save Memory Module(s) to a File

SYNTAX: **save [<opts>] {<modname>}**

FUNCTION: save copies the specified module(s) from memory into your current data directory. The file(s) created in your directory have the same name(s) as the specified module(s).

To save a specified module, type save, followed by the name(s) of the module(s) to save. <modname> must exist in the module directory when saved. The new file is given access permissions for all modes except public write.

If you specify more than one module, each module is stored in a separate file, unless you use the -f option. In that case, all modules listed are saved in the specified file.

NOTE: To save a module, the module must have read access permission for either your group or user ID.

NOTE: save uses the current execution directory as the default directory. Executable modules should generally be saved in the default execution directory.

OPTIONS:

-?	Displays the options, function, and command syntax of save.
-f=<path>	Saves all specified modules to <path>.
-r	Rewrites existing files.
-x	Changes the default directory to the current execution directory.
-z	Reads the module names from standard input.
-z=<file>	Reads the module names from <file>.

EXAMPLES:

```
$ save -x dir copy
$ save -f=/d1/math_pack add sub mul div
```

set — Set Shell Options

SYNTAX: **set [<opts>]**

FUNCTION: set changes shell options for the individual shell in which they are declared.

To change the options for your current shell, enter set and the desired shell options. This command is the equivalent of typing the options directly after the shell prompt on the command line. This is a preferred method of changing shell parameters within procedure files because of its clarity.

The hyphen that usually proceeds declared options is unnecessary when using the set command.

The options specified by set change the shell parameters only in the shell in which they are declared. All descendant shells have the default parameters unless changed within the new shell.

NOTE: set is a built-in shell command. Therefore, it is not in the CMDS directory.

OPTIONS:

?	Displays the options, function, and command syntax of set.
e=<file>	Prints error messages from <file>. If no file is specified, the default file used is /dd/sys/errmsg. Without the -e option, shell prints only error numbers with no message description.
ne	Prints no error messages. This is the default.
l	Must log off system with logout.
nl	Must log off system with <esc>.
p	Displays prompt. The default prompt is $.
np	Does not display prompt.
p=<string>	Sets current shell prompt equal to <string>.
t	Echoes input lines.
nt	Does not echo input lines. This is the default.
v	Verbose mode. Displays a message for each directory searched when executing a command.
nv	Turns off verbose mode.

x	Aborts process upon error. Default.
nx	Does not abort on error.

EXAMPLES: All commands on the same line have the same effect:

$ set x	**$ set -x**	**$ -x**
$ set xp="JOE"	**$ set -xp="JOE"**	**$ -xp="JOE"**

setenv — Set Environment Variables

SYNTAX: **setenv <eparam> <evalue>**

FUNCTION: setenv sets environment variables within a shell for use by the individual shell's child processes.

<eparam> and <evalue> are strings stored in the environment list by shell. These variables are known to the shell in which they are defined and are passed on to descendent processes from that shell.

NOTE: setenv should not be confused with the shell's set command. It has a completely different function. setenv is a built-in shell command. Therefore, it is not in the CMDS directory.

EXAMPLES:

```
$ setenv PATH ../h0/cmds:/d0/cmds:/dd/cmds

$ setenv TERM abm85

$ setenv _sh 0

$ setenv As_long_as_you_want long_value
```

setime — Activate and Set the System Clock

SYNTAX: **setime [<opts>] [y m d h m s [am/pm]]**

FUNCTION: setime sets the system date and time. Once set, it activates the system interrupt clock.

To set the system date and time, type setime, and enter the year, month, day, hour, minute, second, and am or pm as parameters on the command line.

setime does not require field delimiters, but allows you to use the following delimiters between the year, month, day, etc.:

colon (:), semicolon (;), slash (/), comma (,), or space ( )

If semicolons are used as field delimiters, the date and time string must be enclosed by quotes. For example:

```
$ setime "91;1;15;1;25;30;pm"
```

If no parameters are given, setime issues the prompt:

```
$ setime
yy/mm/dd hh:mm:ss [am/pm]
Time:
```

When no am/pm field is specified, OS-9 system time uses the 24 hour clock. For example, 15:20 is 3:20 pm. Midnight is specified as 00:00. Noon is specified as 12:00. Using the am/pm field allows you to use the 12 hour clock. If a conflict exists between the time and the am/pm field (such as 15:20 pm) the system ignores the am/pm designation.

Entering setime echoes the date and time when set.

IMPORTANT NOTE: You must execute this command before OS-9 can perform time-sharing operations. If the system does not have a real-time clock, you should still use this command to set the date for the file system.

Systems with Battery Backed Up Clocks:

setime should still be run to start time-slicing, but you only need to give the -s. The date and time are read from the clock.

OPTIONS:

-?	Displays the options, function, and command syntax of setime.
-d	Does not echo date/time when set.
-s	Reads time from battery backed up clock.

EXAMPLES:

$ setime 91 01 13 15 45	Set to: January 13, 1991, 3:45 PM
$ setime 910113 154500	Same as above
$ setime 91/01/13/3/45/pm	Same as above
$ setime -s	For systems with a battery-backup clock
$ setime **yy/mm/dd hh:mm:ss [am/pm]** **Time:**	No parameters are specified, therefore a prompt is given.

setpr — Set Process CPU Priority

SYNTAX: **setpr <procID> <number>**

FUNCTION: setpr changes the CPU priority of a process.

To use setpr, type setpr, the process ID, and the new priority number of the process to change. setpr may only be used with a process having your ID. The priority number is a decimal number in the range of 1 (lowest) to 65535 (hex FFFF).

Use procs to obtain the ID number and present priority of any current process.

NOTE: This command does not appear in the CMDS directory as it is a built-in shell command.

EXAMPLE: **$ setpr 8 250** Changes the priority of process number 8 to 250.

shell — OS-9 Command Interpreter

SYNTAX: **shell [[set] <arglist>]**

FUNCTION: shell is OS-9's command interpreter program. It reads data from its standard input which is usually the keyboard or a file and interprets the data as a sequence of commands. The basic function of shell is to initiate and control execution of other OS-9 programs.

Usually you enter the shell automatically upon logging into OS-9. The shell displays a dollar sign ($) prompt to show that it is ready and waiting for a command line. You can create a new shell by typing shell optionally followed by a command line.

The shell reads and interprets one text line at a time from standard input. After interpreting each line, the shell reads another line until an end-of-file condition occurs, at which time it terminates itself.

An exception occurs when the shell is called from another program. In this case, the shell processes the specified command as if it was typed on a shell command line. Control returns to the calling program after the single command line is processed. If no command is specified (shell<cr>) or the command is a shell option or built-in command (chd, chx, etc.), more lines are read from standard input and processed as normal. This continues until an end-of-file condition or the logout command is executed.

CAVEAT: The shell's ex command does not recognize utility options unless they are separated from the utility name with a space. For example, ex procs -e works properly, but ex procs-e does not.

The shell uses special characters for various purposes. Special characters consist of the following:

Modifiers:	#	Memory allocation
	^	Process priority modification
	>	Standard output redirection
	<	Standard input redirection
	>>	Standard error output redirection
Separators:	;	Sequential execution
	&	Concurrent execution
	!	Pipe: interprocess communication

Wildcards: * Stands for any string of characters
? Stands for any single character

To send one of these characters to a utility program, you must use a method called ***quoting*** to prevent the shell from interpreting the special character. Quoting consists of enclosing the sequence of characters to be passed to a routine in single or double quotes. For example, '<char>' or "<char>".

The following command line prints the indicated string:

```
$ echo "Hello; goodbye"
Hello; goodbye
```

However, the following command displays the string Hello on your terminal screen and then attempts to execute a program called goodbye.

```
$ echo Hello; goodbye
```

The shell expands the two wildcards to build pathlists. The question mark (?) wildcard matches any single character. The asterisk (*) wildcard matches any string of characters.

dir ???? displays the names of files in the current directory that are four characters long. dir s* displays all names of files in the current directory that begin with s.

Any command that uses a pathlist on the command line accepts a pathlist specified with wildcards. When shell expands the wildcards, if no explicit directory is given, the files in the current data directory are searched for the matched expansion. If an explicit directory name is given in the pathlist, the specified directory is searched.

NOTE: If a command uses an option to search for a file in the current execution directory, wildcards may produce unexpected results. The shell simply reads the current directory or the given relative pathlist containing a wildcard and passes these file names to the command. If the command then tries to find the files relative to the execution directory, the search will most likely fail.

Setting Shell Options

There are two methods of setting shell options. The first method is to type the option on the command line or after the command, shell. For example:

$ -np	Turns off the shell prompt.
$ shell -np	Creates a new shell that does not prompt.

The second method uses the special shell command, set. To set shell options, type set, followed by the options desired. When using set, a hyphen (-) is unnecessary before the letter option. For example:

$ set np	Turns off the shell prompt.
$ shell set np	Creates a new shell that does not prompt.

As you can see, the two methods accomplish the same function. They are both provided for your convenience. Use the method that is clearer for you.

The Shell Environment

For each user on an OS-9 system, the shell maintains a unique list of ***environment*** variables. These variables affect the operation of the shell or other programs subsequently executed. They are ***programmable defaults*** that you can set to your liking.

All environment variables can be accessed by any process called by the environment's shell or descendent shells. This essentially allows you to use the environment variables as ***global*** variables.

NOTE: If a subsequent shell redefines an environment variable, the variable is only redefined for that shell and its descendents.

NOTE: Environment variables are case sensitive.

Several special environment variables are automatically set up when you log on a time-sharing system:

PORT	This specifies the name of the terminal. This is automatically set up by tsmon. /t1 is an example of a legal PORT name.

HOME This specifies your ***home*** directory. The home directory is the directory specified in your password file entry. This is also the directory used when the command chd with no parameters is executed.

SHELL This is the process that is first executed upon logging on to the system.

USER This is the user name you type when prompted by login.

Four other important environment variables are available:

PATH This specifies any number of directories. Each directory must be separated by a colon (:). The shell uses this as a list of commands directories to search when executing a command. If the default commands directory does not include the file/module to execute, each directory specified by PATH is searched until the file/module is found or until the list is exhausted.

PROMPT This specifies the current prompt. By specifying an "at" sign (@) as part of your prompt, you may easily keep track of how many shells you personally have running under each other. The @ is used as a replaceable macro for the shell level number. The environment variable _sh sets the base level.

_sh This specifies the base level for counting the number of shell levels. For example, set the shell prompt to "@howdy: " and _sh to 0:

```
$ setenv _sh 0
$ -p="@howdy: "
howdy: shell
1.howdy: shell
2.howdy: eof
1.howdy: eof
howdy:
```

TERM This specifies the specific terminal being used. This allows word processors, screen editors, and other screen dependent programs to know what type of terminal configuration to use.

The Environment Utilities

Three utilities are available to manipulate environment variables:

- setenv declares the variable and sets its value. The variable is placed in an environment storage area accessed by the shell. For example:

```
$ setenv PATH ..:/h0/cmds:/d0/cmds:/dd/cmds
```

```
$ setenv _sh 0
```

- unsetenv clears the value of the variable and removes it from storage. For example:

```
$ unsetenv PATH
$ unsetenv _sh
```

- printenv prints the variables and their values to standard output. For example:

```
$ printenv
PATH ..:/h0/cmds:/d0/cmds:/dd/cmds
PROMPT howdy
_sh 0
```

The Profile Command

The profile built-in shell command can be used to cause the current shell to read its input from the named file and then return to its original input source, which is usually the keyboard. To use the profile command, enter profile and the name of a file:

```
profile setmyenviron
```

The specified file (in this case, setmyenviron) may contain any utility or shell commands, including commands to set or unset environment variables or to change directories. These changes will remain in effect after the command has finished executing. This is in contrast to calling a normal procedure file by name only. If you call a normal procedure file without using the profile command, the changes would not affect the environment of the calling shell.

Profile commands may be nested. That is, the file itself may contain a profile command for another file. When the latter profile command is completed, the first one will resume.

A particularly useful application for profile files is within a user's .login and .logout files. For example, if each user includes the following line in the .login file, then system-wide commands (common environments, news bulletins, etc.) can be included in the file /dd/SYS/login_sys.

```
profile /dd/SYS/login_sys
```

A similar technique can be used for .logout files.

The Login Shell, .login, and .logout

The ***login shell*** is the initial shell created by the ***login*** program to process the user input commands after logging in. Two special procedure files are extremely useful for personalizing the shell environment:

- .login
- .logout.

To make use of these files, they must be located in your home directory. The .login and .logout files provide a way to execute desired commands when logging on to and leaving the system.

The login shell processes .login as a command file immediately after successful login. This allows you to run a number of initializing commands without remembering each and every command. After processing all commands in the .login file, the shell prompts you for more commands. The main difference in handling the .login file is that the login shell itself actually executes the commands rather than creating another shell to execute the commands. You can issue such commands as set and setenv within the .login file and have them affect the login shell. This is especially useful for setting up the environment variables PATH, PROMPT, TERM, and _sh.

The following is an example .login file:

```
setenv PATH ..:/h0/cmds:/d0/cmds:/dd/cmds:/h0/doc/spex
setenv PROMPT "@what next: "
setenv _sh 0
setenv TERM abm85h
querymail
date
dir
```

.logout is executed when logout is executed to exit the login shell and leave the system. The .logout file is executed before the login shell terminates. Use this to execute any cleaning up procedures that are done on a regular schedule. This might be anything from instigating a backup procedure of some sort to printing a reminder of things to do.

The following is an example .logout file:

```
procs
wait
echo "all processes terminated"
* basic program to instigate backup if necessary *
disk_backup
echo "backup complete"
```

Shell Command Line Syntax

The shell command line consists of a ***keyword*** and optionally any of the parts listed below. The keyword must appear first on a command line. The order of the optional parts depends on the nature of the command and the desired effect. The command line consists of:

Command Line Unit	Description
Keyword	A name of a program or procedure file, a pathlist, or built-in shell command. The shell's built-in commands are:
	ex — Executes a process as overlay.
	chd — Changes your data directory.
	chx — Changes your execution directory.
	kill — Aborts a specified process.
	logout — Terminates the current shell and executes the .logout procedure file if the login shell is terminated.
	profile — Causes the current shell to read its input from the named file and returns to the original input source.
	set — Sets shell options.
	setenv — Sets environment variables.
	setpr — Sets process priority.
	unsetenv — Clears environment variables.
	w — Waits for any one process to finish.
	wait — Waits for all immediate child processes to finish.
Parameter	File or directory names, values, variables, constants, options, etc. to be passed to the program. Wildcards may be used to identify parameter names. The recognized wildcards are:
	* — Matches any character.
	? — Matches any single character.

Execution Modifiers

These modify a program's execution by redirecting I/O or changing the priority or memory allocation of a process:

#<mem size>	Allocates specified memory to a process.
^<priority>	Sets the priority of the process.
<	Redirects standard input.
>[- or +]	Redirects standard output.
>>[- or +]	Redirects standard error output.

The hyphen (-) following the modifiers above signify to write over a specified file. The plus (+) appends the file with the redirected output.

Separators

Separators connect command lines together in the same command line. They specify to the shell how they are to be executed. The separators are:

;	Indicates sequential execution.
&	Indicates concurrent execution.
!	Creates a communication ***pipe*** between processes. Pipes connect the standard output of one process to the standard input of another.

Command Line Execution

The shell command line syntax indicates that a keyword may be a program name, procedure file name, a pathlist, or built-in shell command. Built-in commands are executed immediately by the shell; no directory searching is required, nor is a process created to execute the command. If the specified command is not a built-in command, the shell must locate the program to execute from a number of possible locations. The following procedure describes the actions of the shell when processing a command:

¿ Get command line.

¡ Prepare command:

a. Validate syntax
b. Isolate keyword, parameters, and execution modifiers
c. Expand wildcard names if given

¬ If the keyword is a built-in command, execute the command. Otherwise, search the following directories until the command is found or the directory search is exhausted:

a. The module directory
b. The execution directory
c. Each directory specified by the PATH environment variable

Ð If the command could not be found in the above directories, return error: can't find command.

ƒ If the command is found, load the command into the module directory.

Ý If the load fails, execute shell command (command is assumed to be a procedure file for the shell)

ý If the load succeeds and the module is executable object code, execute command.

« If the load succeeds and the module is BASIC I-code, execute Runb command. Command is an argument for RunB.

» If either of the above command execution fails, return error: can't execute command.

Commands and procedure files in the current execution directory must have the e and/or pe file attribute set or the file will not be found.

If the PATH environment variable is set, its value is interpreted as a list of directories to search if the initial search of the execution directory fails. If an absolute pathlist, a path beginning with a slash (/) is given as the command, the shell does not perform the PATH directory search. The following are examples of setting up the PATH variable:

```
setenv PATH /d0
setenv PATH /h0/cmds:/n0/jack/h0/cmds:/n0/jill/h0/cmds
setenv PATH kim:../kim:.../cmds
```

Each directory name is separated by a colon (:). Shell isolates the directory name and appends it to the command name and uses this pathlist to load the command. If the load fails, the next directory given is used until the command is successfully loaded or all directories are tried. Regardless of the error encountered, the shell continues with the next directory. If a directory given is a relative pathlist, the pathlist is relative to the execution directory.

To assist in determining the directory from which a command was loaded or not loaded, turn on the -v option to display the shell's progress while searching the directories.

The login program automatically sets the PATH variable to the execution directory from which login itself was loaded if the password entry gives an execution directory

other than ".". The period (.) tells the shell to use the login's execution directory.

Example Command Lines

The following example displays a numbered listing of the data directory. dir is a keyword indicating the dir utility. -u is a parameter for dir. The exclamation point (!) is a pipe that redirects the unformatted output of dir to the standard input of pr. pr is a keyword indicating the pr utility. -n is a parameter for pr.

```
dir -u ! pr -n
```

The following command line lists all files in the current data directory that have names beginning with s. list is the keyword. s* identifies the parameters.

```
list s*
```

update uses master as standard input in this next example. The output from update is used as input for sort. The output from sort is redirected to the printer.

```
update <master ! sort >/p1
```

OPTIONS:

-?	Displays the options, function, and command syntax of shell.
-e=<file>	Prints error messages from <file>. If no file is specified, the default file used is /dd/sys/errmsg. Without the -e option, shell prints only error numbers with no message description.
-ne	Prints no error messages. This is the default.
-l	The logout built-in command is required to terminate the login shell. <eof> does not terminate the shell.
-nl	<eof> terminates the login shell. The <Esc> key normally sends an <eof> to the shell.
-p	Displays prompt. The default prompt is $.
-np	Does not display prompt.

-p=<string>	Sets current shell prompt equal to <string>.
-t	Echoes input lines.
-nt	Does not echo input lines. This is the default.
-v	Verbose mode. Displays a message for each directory searched when executing a command.
-nv	Turns off verbose mode.
-x	Aborts process upon error. This is the default.
-nx	Does not abort on error.

sleep — Suspend a Process for a Period of Time

SYNTAX: **sleep [<opts>] <num>**

FUNCTION: sleep puts your process to sleep for a number of ticks. It is generally used to generate time delays in procedure files.

To use the sleep utility, type sleep, followed by the number of ticks you want the process to sleep. A tick count of one causes the process to give up its current time slice and return immediately. A tick count of zero causes the process to sleep indefinitely, usually until awakened by a signal. The duration of a tick is system-dependent.

sleep is generally used to generate time delays in procedure files.

OPTIONS:

-? Displays the option and command syntax of sleep.

-s Changes count representation to seconds.

NOTE: Only one option may be used on the command line. If not specified, <num> defaults to zero.

EXAMPLES:

$ sleep 25	Sleep for 25 ticks.
$ sleep -s 1000	Sleep for 1000 seconds.

tape — Tape Controller Manipulation

SYNTAX: **tape {<opts>} [<dev>]**

FUNCTION: tape provides a means to access a tape controller from a terminal. tape can rewind, erase, skip forwards and backwards, and write tapemarks to a tape.

If the tape device <dev> is not specified on the command line and the -z option is not used, tape uses the default device /mt0.

OPTIONS:

-?	Displays options, function, and command syntax of tape.
-b[=<num>]	Skips the specified number of blocks. The default is one block. If <num> is negative, the tape skips backward.
-e=<num>	Erases a specified number of blocks of tape.
-f[=<num>]	Skips the specified number of tapemarks. The default is one tapemark. If <num> is negative, the tape skips backward.
-o	Puts tape off-line.
-r	Rewinds the tape.
-s	Determines the block size of the device.
-t	Retensions the tape.
-w[=<num>]	Writes a specified number of tapemarks. The default is one tapemark.
-z	Reads a list of device names from standard input. The default device is /mt0.
-z=<file>	Reads a list of device names from <file>.

If you specify more than one option, tape executes each option function in a specific order. Therefore, you can skip ahead a specified number of blocks, erase, and then rewind the tape all with the same command.

The order of option execution is as follows:

- ¿ Gets device name(s) from the -z option.
- ¡ Skips the number of tapemarks specified by the -f option.
- ¬ Skips the number of blocks specified by the -b option.
- Ð Writes a specified number of tapemarks.
- *f* Erases a specified number of blocks of tape.
- Ý Rewinds the tape.
- ý Puts the tape off-line.

EXAMPLES:

$ tape /mt0 -r	Rewinds tape on device /mt0.
$ tape -f=5 -e=2 -r	Skips forward five files on device /mt0, erases the next two blocks, and then rewinds the tape.

tapegen — Put Files on a Tape

SYNTAX: **tapegen [<opts>] <filename> <filename>**

FUNCTION: tapegen creates the "bootable" tape. tapegen is a standard utility that performs a function similar to the os9gen utility. Both utilities place the bootstrap file on the media and mark the media identification block with information regarding the bootstrap file. In addition, tapegen can optionally place initialized data on the tape, for application-specific purposes.

To use the tapegen utility, type tapegen followed by any desired options.

OPTION:

-?	Displays the options, function, and command syntax of tapegen.
-b=<bootfile>	Installs an OS-9 boot file.
-bz	Reads boot module names from standard input.
-bz=<bootlist>	Reads boot module names from the specified bootlist file.
-c	Checks and displays header information.
-d=<dev>	Specifies the tape device name. The default is /mt0.
-o	Takes the tape drive off-line when finished.
-t=<target>	Specifies the name of the target system.
-i=<file>	Installs an initialized data file on the tape. This is usually a RAM disk image.
-v=<volume>	Specifies the name of the tape volume.
-z	Reads filenames from standard input.
-z=<file>	Reads filenames from the specified file.

EXAMPLES: The following example makes a bootable tape. The disk image is derived from the /dd device.

```
$ tapegen -b=OS9Boot.tape -i=/dd@ "-v=OS-9/68K Boot Tape" -t=MySystem
```

This example makes a bootable tape with no initialized data file. The "header" information is displayed after writing the tape.

```
$ tapegen -b=OS9Boot.h0 -c
```

tee — Copy Standard Input to Multiple Output Paths

SYNTAX: **tee {<path>}**

FUNCTION: tee is a filter that copies all text lines from its standard input to its standard output and any other additional pathlists given as parameters.

To use the tee utility, type tee and the pathlist(s) to which standard input is to be redirected. This utility is generally used with input redirected through a pipe.

OPTION: -? Displays the function and command syntax of tee.

EXAMPLES: The example below uses a pipeline and tee to simultaneously send the output listing of dir to the terminal, printer, and a disk file:

```
$ dir -e ! tee /printer /d0/dir.listing
```

This example sends the output of an assembler listing to a disk file and the printer:

```
$ asm pgm.src l ! tee pgm.list >/printer
```

This example broadcasts a message to three terminals:

```
$ echo WARNING System down in 10 minutes ! tee /t1 /t2 /t3
```

tmode — Change Terminal Operating Mode

SYNTAX: **tmode [<opts>] [<arglist>]**

FUNCTION: tmode displays or changes the operating parameters of your terminal.

NOTE: tmode can only be used for SCF or GFM devices.

To change the operating parameters of your terminal, type tmode and any parameters you want changed. If no parameters are given, the present values for each parameter are displayed. Otherwise, the parameter(s) given in the parameter list are processed. You can give any number of parameters, separated by spaces or commas.

If a parameter is set to zero, OS-9 no longer uses the parameter until it is re-set to a recognizable code. For example, to set xon and xoff to zero, type:

```
tmode xon=0 xoff=0
```

Consequently, OS-9 does not recognize xon and xoff until the values are re-set.

To re-set the value of a parameter to its default, type tmode and specify the parameter with no value. This re-sets the parameter to the default value given in this manual.

Use the -w=<path#> option to specify the path number to be affected. If none is given, standard input is affected.

NOTE: If you use tmode in a shell procedure file, you must use the option -w=<path#> to specify one of the standard paths (0, 1, or 2) to change the terminal's operating characteristics. The change remains in effect until the path is closed. For a permanent change to a device characteristic, you must change the device descriptor. You may alter the device descriptor to set a device's initial operating parameters using xmode. See the xmode utility for more information.

You cannot change the following five parameters by tmode: type, par, cs, stop, and baud. These are included in tmode for informational purposes only. You can only change these by altering the device descriptor and using iniz. See xmode for more information.

tmode can work only if a path to the file/device has already been opened. The ***OS-9 Technical Manual*** contains full information on device descriptors.

Tmode Parameter Names

Name	Function
upc	Upper case only. Lower case characters are converted automatically to

upper case.

noupc	Upper and lower case characters permitted. Default.
bsb	Erase on backspace. Backspace characters are echoed as a backspace-space-backspace sequence. Default.
nobsb	No erase on backspace. Echoes single backspace only.
bsl	Backspace over line. Lines are deleted by sending backspace-space-backspace sequences to erase the same line for video terminals. Default.
nobsl	No backspace over line. Lines are deleted by printing a new line sequence for hard-copy terminals.
echo	Input characters echoed back to terminal. Default.
noecho	No echo
lf	Auto line feed on. Line feeds are automatically echoed to terminal on input and output carriage returns. Default.
nolf	Auto line feed off
null=n	Set null count. Number of null ($00) characters transmitted after carriage returns for return delay. The number is decimal. The default null count is 0.
pause	Screen pause on. Output suspended upon full screen. See pag parameter for definition of screen size. Output can be resumed by typing any key.
nopause	Screen pause mode off
pag=n	Set video display page length to n lines, where n is in decimal. Used for pause mode, see above.
bsp=h	Set input backspace character (normally <control>H, default = 08). Numeric value of character in hexadecimal.
del=h	Set input delete line character (normally <control>X, default = 18). Numeric value of character in hexadecimal.

Name	**Function**
eor=h	Set end-of-record input character (normally <cr>, default = 0D). Numeric value of character in hexadecimal.

eof=h Set end-of-file input character (normally <esc>, default = 1B). Numeric value of character in hexadecimal.

reprint=h Set reprint line character (normally <control>D, default = 04). Numeric value of character in hexadecimal.

dup=h Sets the duplicate last input line character (normally <control>A, default = 01). Numeric value of character in hexadecimal.

psc=h Set pause character (normally <control>W, default = 17). Numeric value of character in hexadecimal.

abort=h Abort character (normally <control>C, default = 03). Numeric value of character in hexadecimal.

quit=h Quit character (normally <control>E, default = 05). Numeric value of character in hexadecimal.

bse=h Set output backspace character (default = 08). Numeric value of character in hexadecimal.

bell=h Set bell (alert) output character (default = 07). Numeric value of character in hexadecimal.

type=h ACIA initialization value: shows parity, character size, and number of stop bits. Value in hexadecimal. This value is affected by changing the individual par(ity), cs (character length), and stop (stop bits) values. This value cannot be changed by tmode.

par=s Shows parity using one of the following strings: odd, even, or none. Changing parity will affect the type value. Parity cannot be changed by tmode.

cs=n Shows character length using one of the following values:

n = 8, 7, 6 or 5 (bits)

Changing character length will change the type value. Character length cannot be changed by tmode.

Name	Function

stop=n Shows number of stop bits used:

n = 1, 1.5 or 2 (stop bits)

Changing the stop bit value affects the type value. The number of stop bits used cannot be changed by tmode.

baud=n — Baud rate: The baud rate may currently be set to the following values:

n = **50**	**134.5**	**600**	**2000**	**4800**	**19200**
75	**150**	**1200**	**2400**	**7200**	**38400**
110	**300**	**1800**	**3600**	**9600**	**extern**

The baud rate cannot be changed by tmode.

xon=h — DC1 resume output character (normally <control>Q, default = 11). Numeric value of character in hexadecimal.

xoff=h — DC2 suspend output character (normally <control>S, default = 13). Numeric value of character in hexadecimal.

tabc=h — Tab character (normally <control>I, default = 09). Numeric value of character in hexadecimal.

tabs=n — Number of characters between tab stops. The number is in decimal. The default is 4 characters between tab stops.

normal — Set the terminal back to its default characteristics. This will not affect the following values: type, baud rate, parity, character length, and stop bits.

OPTIONS:

-? — Displays the option, function, and command syntax of tmode.

-w=<path#> — Changes the path number <path#> affected.

EXAMPLES:

```
$ tmode noupc lf null=4 bse=1F pause

$ tmode pag=24 pause bsl noecho bsp=8 bsl=C

$ tmode xon xoff quit=5
```

touch — Update the Last Modification Date of a File

SYNTAX: **touch [<opts>] {<path>}**

FUNCTION: touch updates the last modification date of a file. Usually, this command is used with a make command's ***makefile***. Associated with every file is the date the file was last modified. touch simply opens a file and closes it to update the time the file was last modified to the current date.

To update the last modification date of a file, type touch and the pathlist of the file to update. touch searches the current data directory for the file to update if another directory or the -x option is not specified.

NOTE: If the specified file is not found, touch creates a file with a current modification date.

OPTIONS:

-?	Displays the options, function, and command syntax of touch.
-c	Does not create a file if not found.
-q	Does not quit if an error occurs.
-x	Searches the execution directory for the file.
-z	Reads the file names from standard input.
-z=<path>	Reads the file names from <path>.

EXAMPLES:

```
$ touch -c /h0/doc/program

$ touch -cz

$ dir -u ! touch
```

tr — Transliterate Characters

SYNTAX: **tr [<opts>] <str1> [<str2>] [<path1>] [<path2>]**

FUNCTION: tr transliterates characters from <str1> into a corresponding character from <str2>. If <str1> contains more characters than <str2>, the final character in <str2> is used for each excess character in <str1>.

To use the tr utility, type tr and the characters to search for (<str1>), and optionally, the replacement characters (<str2>), the input file's pathlist (<path2>) and the output file's pathlist (<path2>).

<str1> is required. If <str2> is missing, all characters in <str1> are deleted from the output. If <path1> and <path2> are missing, standard input and output are assumed. If only one path is specified, it is used as the input file pathlist.

<str1> and <str2> are interpreted as character classes. To facilitate creating character classes, use the following metacharacters:

Char	Name/Description
-	**RANGE**. The hyphen (-) is defined as representing all characters lexicographically greater than the preceding character and less than the following character. For example: [a-z] is equivalent to the string abcdefghijklmnopqrstuvwxyz. [m-pa-f] is equivalent to the string mnopabcdef. [0-7] is equivalent to the string 01234567. See the ASCII chart in Appendix A for character values.

Char	Name/Description
\	**ESCAPE**. The backslash (\) removes special significance from special characters. It is followed by a base and a numeric value or a special character. If no base is specified, the base for the numeric value defaults to hexadecimal. An explicit base of decimal or hexadecimal can be specified by preceding the numeric value with a qualifier of d or x, respectively. It also allows entry of some non-printing characters such as: \t = Tab character \n = New-line character \l = Line feed character \b = Backspace character \f = Form feed character

NOTE: Do not confuse <str1> and <str2> with the ***character class*** regular expression. <str1> and <str2> do not need surrounding brackets. Brackets are merely treated as characters in the character class.

OPTIONS:

-?	Displays the options, function, and command syntax of tr.
-c	Transliterates all ASCII characters (1 through $7F) to <str2>, except for the set of characters in <str1>.
-d	Deletes all matching input characters and expressions.
-s	Squeezes all repeated output characters or expressions in <str2> to single characters or expressions.
-v	Same as -c.
-z	Reads standard input for list of file names.
-z=<path>	Reads the file names from <path>.

You can generally give options anywhere on the command line. If you wish to use the pathlists but not <str2>, you must specify the -d option prior to the pathlists. Similarly, if you use the -z option to read pathlists from standard input, the -z must precede <path2>.

The -s option does not differentiate between characters originally in <str2> and transliterated characters. It always returns a string with no consecutively repeated characters. For example, the command tr -s abcde x transliterates the string exasperate into xspxrxtx.

The -s and -d options are mutually exclusive.

If you use the -c option to change all but a certain sequence of characters, it also changes carriage returns and newlines unless they are specified in the sequence of characters.

WARNING: tr always deletes ASCII nul ($00).

EXAMPLES: The following examples use standard input for the input to tr. The output is sent to standard output. Thus, the first line following each command line is the standard input, and the second line is the standard output.

```
$ tr abcd jklm
aabdc_efg
jjkml_efg
```

Transliterates standard input, converting a to j, b to k, c to l, and d to m.

```
$ tr abcd j
abcd_efgh
jjjj_efgh
```

Transliterates standard input, converting each a, b, c, and d to j.

```
$ tr a-d k
abc_abcd-efgh
kkk_kkkk-efgh
```

Transliterates standard input, converting each character contained in the expression abcd to k.

```
$ tr abcd
abcd_efgh
_efgh
```

Transliterates standard input, deleting each a, b, c, and d.

```
$ tr -d abcd
abcdefg
efg
```

Transliterates standard input, deleting each a, b, c, and d.

```
$ tr -s dcba eocd
edenbcada
encoded
```

Transliterates standard input converting d to e, c to o, b to c, and a to d. Consecutively repeated output characters, the matching eocd, are squeezed into a single character

```
$ tr -c a-zA-Z \n
one word per line
one
word
per
line
```

Transliterates standard input, converting all non-alphabetic characters to newline characters.

tsmon Supervise Idle Terminals and Initiate the Login Command

SYNTAX: **tsmon [<opts>] {/<dev>}**

FUNCTION: tsmon supervises idle terminals and starts the login utility in a timesharing application. Typically, tsmon is executed as part of the start-up procedure when the system is first brought up and remains active until the system shuts down.

The parameter /<dev> specifies a terminal to monitor. This is generally an SCF device.

You can specify up to 28 device name pathlists for tsmon to monitor. When you type a carriage return on any of the specified paths, tsmon automatically forks login, with standard I/O paths opened to the device. If login fails because you could not supply a valid user name or password, control returns to tsmon.

Most programs terminate when an end-of-file character (normally <escape>) is entered as the first character on a command line. This logs you off the system and returns control to tsmon.

tsmon prints a message when you log off:

Logout after 11 minutes, 30 seconds. Total time 3:57:46.

The Total time figure is the total amount of time that the terminal has accumulated on-line since the tsmon was started.

tsmon is normally used to monitor I/O devices capable of bi-directional communication, such as CRT terminals. However, you may use tsmon to monitor a named pipe. If this is done, tsmon creates the named pipe, and then waits for data to be written to it by some other process.

When data arrives, tsmon starts a shell with its input redirected to the pipe file. This is useful for starting remote processes in a networked environment.

You can run several tsmon processes concurrently, each one watching a different group of devices. This must be done when more than 28 terminals are to be monitored, but is sometimes useful for other reasons. For example, you may want to keep modems or terminals suspected of hardware trouble isolated from other devices in the system.

tsmon forks login with the PORT environment variable set to the SCF device name and all other environment variables cleared.

OPTIONS:

-? Displays the options, function, and command syntax of tsmon.

-d Displays statistics when a ^\ character (control-backslash or hex $1C) is typed on a monitored terminal.

-l=<prog>	Forks <prog>, an alternate login program.
-p	Displays an "online" prompt to each timesharing terminal being monitored by tsmon.
-r=<prog>	Forks an alternate shell program.
-z	Reads the device names from standard input.
-z=<path>	Reads the device names from <path>.

EXAMPLES: This command starts timesharing on term and t1, printing a welcome message to each. A similar command might be used as the last line of a system startup file.

```
tsmon -dp /term /t1&
2 devices online                (confirmation by tsmon)
```

The -d option causes tsmon to print various statistics about the devices being monitored whenever control-backslash (^\) is typed on either terminal. The statistics might look something like this:

```
tsmon started 12-11-90 20:38:15 with 2 devices     0:36:06
 /term     quiet at  0:08:07   cumulative time  3:29:30   logins: 1/9
*/t1       quiet at  0:36:03   cumulative time  3:57:46   logins: 2/4
```

NOTE: The standard input device shown for tsmon by the procs utility always indicates the last device to gain tsmon's attention.

You must implement the SS_SSig I$SetStat function (send signal on data ready) on any device to be monitored by tsmon. Because this function is used (for example, instead of I$ReadLn), it is possible to output data to a terminal that is *not* logged in without having to wait for someone to press a key.

unlink — Unlink Memory Module

SYNTAX: **unlink [<opts>] {<modname>}**

FUNCTION: unlink tells OS-9 that you no longer need the memory module(s) named

To unlink an attached module, type unlink and the name(s) of the module(s) to unlink. The link count is then be decremented by one. If the link count becomes zero, the module directory entry is deleted and the memory is de-allocated. It is good practice to unlink modules whenever possible to make most efficient use of available memory resources.

WARNING: Never unlink a module you did not link to or load. Unlinking a module more than once may prematurely lower its link count and possibly destroy the module while it is still in use.

OPTIONS:

-?	Displays the options, function, and command syntax of unlink.
-z	Reads the module names from standard input.
-z=<file>	Reads the module names from <file>.

EXAMPLES:

$ unlink pgm pm5 pgm9

Unlinks pgm, pgm5, and pgm9 and lowers the link count of each module by one.

$ dir -u ! unlink -z

Pipes an unsorted listing of the current data directory to unlink. This unlinks all modules contained in the directory and lowers the link count of each module by one.

$ unlink -z=namefile

Unlinks each module listed in namefile and lowers the link count of each module by one.

```
$ mdir
  Module Directory at 14:44:35
kernel    init     p32clk    rbf      p32hd
h0        d0       r0        edit     mdir
$ unlink edit
$ mdir
  Module Directory at 14:44:35
kernel    init     p32clk    rbf      p32hd
h0        d0       r0        mdir
```

unsetenv — Clear Environment Parameter

SYNTAX: **unsetenv <eparam>**

FUNCTION: unsetenv deletes the specified environment variable from the environment list.

To use setenv, type unsetenv, followed by the environment parameter to delete. This removes the variable from the environment list.

NOTE: If the specified variable has not been previously defined, unsetenv has no effect and it gives you no message.

EXAMPLES:

```
$ unsetenv _sh
$ unsetenv TERM
```

SEE ALSO: setenv and printenv utility descriptions

w/wait — Wait for One/All Child Process(es) to Terminate

SYNTAX:

```
w
wait
```

FUNCTION: w causes the shell to wait for the termination of one child process before returning with a prompt. wait causes the shell to wait for all child processes to terminate before returning with a prompt.

Type w or wait and a carriage return. When the shell prompt is displayed, the child process(es) have terminated.

EXAMPLES:

```
$ list file1 >/p1&
$ list file2.temp ! filter >file2&
$ wait
$ list file2 >/p1
```

In this example, the prompt returns when the first of these three processes (one, two, or three) terminates:

```
$ one&
$ two&
$ three$
$ w
$
```

xmode — Examine or Change Device Initialization Mode

SYNTAX: **xmode [<opts>] <devname> [<arglist>] {<devname>}**

FUNCTION: xmode displays or changes the initialization parameters of any SCF-type device such as a video display, printer, RS-232 port, etc. Some common uses are to change the baud rates and control key definitions.

NOTE: xmode can only be used for SCF or GFM devices.

To use the xmode utility, type xmode and any parameters to change. If no parameters are given, the present values for each parameter are displayed. Otherwise, the parameter(s) given in the parameter list are processed. You can give any number of parameters, separated by spaces or commas. You must specify a device name to process the parameter(s) given in the parameter list.

If a parameter is set to zero, the device no longer uses the parameter until it is re-set to a recognizable code. For example, set xon and xoff to zero:

```
xmode /term xon=0 xoff=0
```

/term will not recognize xon and xoff until the values are re-set.

To re-set the values of a parameter to its default, type xmode and specify the parameter with no value. This re-sets the parameter to the default value given in this manual.

xmode is similar to the tmode utility. tmode only operates on open paths so it has a temporary effect. xmode actually updates the device descriptor. The change persists as long as the computer is running, even if paths to the device are repetitively opened and closed.

Five parameters need further explanation: type, par, cs, stop, and baud. These parameters are changed by xmode only if the device is iniz-ed directly after the xmode changes are made. This is usually done in the startup file or by first deiniz-ing a file. For example, the following command sequence changes the baud rate of /t1 to 9600:

```
$ deiniz t1
$ xmode baud=9600
$ iniz t1
```

This type of command sequence changes the device descriptor and initializes it on the system. Only the five parameters mentioned above need this special sequence changed. All other xmode parameters are changed immediately.

OPTIONS: -? Display the options, function, and command syntax of xmode.

-z Reads device names from standard input.

-z=<file> Reads device names from <file>.

Xmode Parameter Names

Name	Function
upc	Upper case only. Lower case characters are converted automatically to upper case.
noupc	Upper and lower case characters are permitted. Default.
bsb	Erase on backspace. Backspace characters are echoed as a backspace-space-backspace sequence. Default.
nobsb	No erase on backspace. Echoes single backspace only.
bsl	Backspace over line. Lines are deleted by sending backspace-space-backspace sequences to erase the same line for video terminals. Default.
nobsl	No backspace over line. Lines are deleted by printing a new line sequence for hard-copy terminals.
echo	Input characters echoed back to terminal. Default.
noecho	No echo
lf	Auto line feed on. Line feeds are automatically echoed to terminal on input and output carriage returns. Default.
nolf	Auto line feed off
null=n	Set null count. Number of null ($00) characters transmitted after carriage returns for return delay. The number is decimal. By default, the null count is set to zero.
pause	Screen pause on. Output suspended upon full screen. See pag parameter for definition of screen size. Output can be resumed by typing any key.
nopause	Screen pause mode off
pag=n	Set video display page length to n lines. n is a decimal number. Used for pause mode, see above.

Name	Function
bsp=h	Set input backspace character (normally <control>H, default = 08).

Numeric value of character in hexadecimal.

del=h	Set input delete line character (normally <control>X, default = 18). Numeric value of character in hexadecimal.
eor=h	Set end-of-record input character (normally <cr>, default = 0D). Numeric value of character in hexadecimal.
eof=h	Set end-of-file input character (normally <esc>, default = 1B). Numeric value of character in hexadecimal.
reprint=h	Set reprint line character (normally <control>D, default = 04). Numeric value of character in hexadecimal.
dup=h	Set duplicate last input line character (normally <control>A, default = 01). Numeric value of character in hexadecimal.
psc=h	Set pause character (normally <control>W, default = 17). Numeric value of character in hexadecimal.
abort=h	Abort character (normally <control>C, default = 03). Numeric value of character in hexadecimal.
quit=h	Quit character (normally <control>E, default = 05). Numeric value of character in hexadecimal.
bse=h	Set output backspace character (default = 08). Numeric value of character in hexadecimal.
bell=h	Set bell (alert) output character (default = 07). Numeric value of character in hexadecimal.
type=h	ACIA initialization value. Sets parity, character size, and number of stop bits. Value in hexadecimal. This value is affected by changing the individual par(ity), cs (character length), and stop (stop bits) values. This value is not affected by the xmode normal command. This value is not changed until the specified device is iniz-ed.
par=s	Sets parity using one of the following strings: odd, even, or none. Setting parity affects the type value. This value is not affected by xmode normal. This value is not changed until the specified device is iniz-ed.

Name	Function
cs=n	Sets character length using one of the following values:

n = 8, 7, 6 or 5 (bits)

Setting character length changes the type value. This value is not affected by xmode normal. This value is not changed until the specified device is iniz-ed.

stop=n Sets the number of stop bits used:

n = 1, 1.5 or 2 (stop bits)

Setting the stop bit value affects the type value. This value is not affected by xmode normal. This value is not changed until the specified device is iniz-ed.

baud=n Baud rate. The baud rate may currently be set to the following values:

n = 50 134.5 600 2000 4800 19200
75 150 1200 2400 7200 38400
110 300 1800 3600 9600 extern

This value is not affected by xmode normal and is not changed until the specified device is iniz-ed.

xon=h DC1 resume output character (normally <control>Q, default = 11). Numeric value of character in hexadecimal.

xoff=h DC2 suspend output character (normally <control>S, default = 13). Numeric value of character in hexadecimal.

tabc=h Tab character (normally <control>I, default = 09). Numeric value of character in hexadecimal.

tabs=n Number of characters between tab stops. The number is in decimal. By default, there are four characters between tab stops.

normal Set the terminal back to its default characteristics. This does not affect the following values: type, baud rate, parity, character length, and stop bits.

EXAMPLES: **$ xmode /term noupc lf null=4 bse=1F pause**

$ xmode /t1 pag=24 pause bsl noecho bsp=8 bsl=C

NOTES

End of Chapter