Command

COHERENT

Command Manual

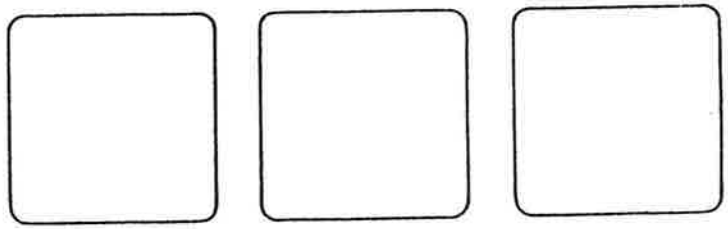

Table of Contents

Table of Contents

Introduction

This manual contains a concise description of each command available to a user of the COHERENT™ operating system. It is a reference work rather than a tutorial; its descriptions use terminology unfamiliar to the novice user. The novice should read the *Introduction to the COHERENT System* first.

This manual consists of many sections, ordered alphabetically. Each section describes a single command; this manual grows as new commands become available on the system. Command names are usually short and sometimes obscure to the uninitiated; the **Index** provides functional descriptions.

Commands typed to the COHERENT system are usually interpreted by the command line interpreter, or *shell*, described in section **sh** of this manual. The *sh Command Language Tutorial* gives further information on the shell. Section **stty** describes characters with special meanings and other terminal characteristics.

The **USAGE** line of each section describes how to invoke the command. For example, the **USAGE** of the **cat** command is:

cat [−u] [*file* **...]**

Boldface type indicates characters which are actually typed when invoking the command. Italic type indicates parts of the command line which are actually replaced by other text when invoking the command. Square brackets '[' and ']' indicate parts of the command line which are optional. Ellipses "..." indicate that the preceding part of the command line may be repeated. The command

cat a b c

is an example of the **USAGE** of **cat**, with the optional **−u** omitted and **a b c** specifying the desired *file* ... of the command. Command options are normally indicated by optional *flags* on the command line, such as **−u** above. The notation **[−abcd]** indicates that any or all of the flags **abcd** may be specified.

Many sections include a **FILES** subsection describing the names of files used by the command. Sometimes a pathname includes a shell variable such as **$HOME**, which is actually a variable containing a file name rather than a literal file name.

Most sections include cross references in a **SEE ALSO** subsection. Many refer to other sections of this volume. Others refer to the *COHERENT System Manual*, which provides detailed information on the basic building blocks of the COHERENT system. Some refer to separate documents describing complicated commands in detail.

Commands normally return an integer status. One byte of the status indicates how the system terminated the command—for example, normal termination or termination on receipt of a signal. If a command terminated normally, the other status byte, called the *exit status* of the command, usually indicates the success (zero) or failure (nonzero) of the command. The **DIAGNOSTICS** subsection describes error messages and the exit status of a command. The exit status may be manipulated by commands to the shell **sh**.

Several sections include the heading "Maintenance". These commands are generally used by the system administrator rather than by most users. Some require special permission for execution. Section **Maintenance** lists sections which describe maintenance commands. The *COHERENT Administrator's Guide* gives more information on system maintenance and the *COHERENT System Manual* describes a few additional maintenance commands.

Sections with the heading "Games" describe games which are distributed with the COHERENT system. Section **Games** lists sections which describe games.

The COHERENT system is available on a wide variety of computers. In almost all cases, the operation of a COHERENT command is identical on different machines; the system looks the same to the user, regardless of which processor actually executes commands. However, because of hardware limitations, a few commands described in this manual may not exist or may not work as expected on some systems. A document describing divergences from the descriptions in this manual and also describing additional system-dependent commands is provided with such systems. In particular, commands requiring large data files may not work as expected on systems with small disk file systems, simply because the disk space available makes it impossible to include the necessary files.

This manual is also available online, through the **man** command. The **help** command prints a more concise description of command usage. For example,

man cat

prints the entire section of this manual describing **cat**, while

help cat

prints a brief summary. Because the files for the **man** command are quite large, they might not be included on COHERENT systems with insufficient disk space. **help** information is available on all COHERENT systems.

Unless otherwise noted in the **USAGE** section, commands described in this manual are found in directories **/bin** or **/usr/bin** on the COHERENT system. Maintenance commands are often found in directory **/etc**, and games are usually in directory **/usr/games**. The shell **sh** executes a few commands directly.

Games

The following commands are games, indicated with the heading "Games" in this manual.

fortune
rubik

Maintenance

The following commands are primarily system maintenance commands, indicated with the heading "Maintenance" in this manual. These commands are generally used by the system administrator. The *COHERENT Administrator's Guide* gives further information on the use of these commands. The *COHERENT System Manual* describes a few additional maintenance commands.

ac
accton
bad
check
clri
dcheck
dump
dumpdate
dumpdir
icheck
load
mkfs
mknod
mount
ncheck
newusr
quot
restor
sa
sync
uload
umount
wall

NAME
ac – summarize login accounting information

USAGE
ac [–dp] [–w *wfile* **] [** *username ...* **]**

DESCRIPTION
One of the accounting mechanisms available on the COHERENT system is login accounting, which keeps track of the time each user spends logged into the system. Creating the file **/usr/adm/wtmp** enables login accounting. Subsequently **date**, **login**, and **init** (*COHERENT System Manual*) write raw accounting data to the file to record the time, terminal name and user name for each date change, login, logout, or system reboot.

ac summarizes the raw accounting data. By default, **ac** prints the total connect time found in **/usr/adm/wtmp**. Any *username* restricts the summary to each specified user.

The available options are:

–d Itemize the output into daily (midnight to midnight) periods.

–p Print individual totals.

–w Use *wfile* rather than **/usr/adm/wtmp** as the raw data file.

FILES
/usr/adm/wtmp

SEE ALSO
date, login, sa
COHERENT System Manual: **init, utmp.h**
COHERENT Administrator's Guide

NOTES
As the file **/usr/adm/wtmp** can become very large, it should be truncated periodically. Special care should be taken if login accounting is enabled on a COHERENT system with a small disk file system.

Maintenance

NAME
accton – enable/disable process accounting

USAGE
/etc/accton [*file*]

DESCRIPTION
One of the accounting mechanisms available on the COHERENT system is *process accounting*, also called *shell accounting*. Process accounting keeps track of each process executed by each user.

accton with a *file* argument enables process accounting. Subsequently, the system writes raw accounting data into *file*, which normally should be **/usr/adm/acct**. The **sa** command summarizes the resulting raw statistics.

With no arguments, **accton** disables process accounting.

accton is normally invoked by the initialization command file **/etc/rc**.

FILES
/usr/adm/acct raw accounting data

SEE ALSO
ac, sa
COHERENT System Manual: **acct, acct.h, init**
COHERENT Adminstrator's Guide

NOTES
As the file **/usr/adm/acct** can become very large, special care should be taken if process accounting is enabled on a COHERENT system with a small disk file system.

NAME
ar – maintain archives

USAGE
ar *option*[*modifier*] [*position*] *archive* [*member* ...]

DESCRIPTION
ar combines several object files into a file called an *archive* or a *library*. Archives reduce the size of directories and allow many files to be handled as a single unit. The linker **ld** understands the archive format. Each *member* of an *archive* is an object file, identified by the last component of its pathname.

The mandatory *option* argument consists of one of the following command keys:

d Delete each given *member* from the *archive*.

m Move each given *member* in the *archive*. If no *modifier* is given, move each *member* to the end.

p Print each *member*.

q Quick append. Append each *member* to the end of the *archive* unconditionally.

r Replace each *member* of the *archive*. The following *modifier* specifies how to perform the replacement, as described below.

t Print a table of contents listing each *member* specified. If none is given, list all in the *archive*. The modifier **v** provides additional information.

x Extract each given *member* and place in the current directory. If none is specified, extract all members. The *archive* is not changed.

The *modifier* may be one of the following. The modifiers **abiu** may be used only for the **m** and **r** options.

a If the new *member* does not already exist in the *archive*, insert it after the given *position*.

b If the new *member* does not already exist in the *archive*, insert it before the given *position*.

c Suppress the message normally printed when **ar** creates an *archive*.

i If the new *member* does not already exist in the *archive*, insert it in front of the given *position* (same as **b**).

l Create temporary files in the current directory instead of in **/tmp**.

u Update the *archive* only if the *member* is newer than the version in the *archive*.

v Generate verbose messages.

Archives are canonized, as described under **canon.h** in the *COHERENT System Manual*.

FILES
/tmp/v* temporary

SEE ALSO
ld, ranlib
COHERENT System Manual: **ar.h, canon.h**

NAME
as, i8086as, m68000as, pdp11as, z8001as, z8002as – assembler

USAGE
as [**–glx**] [**–o** *ofile*] *file* ...

DESCRIPTION
as is the assembler for the host system. Cross assemblers for other machines are named *xxxxxx***as**, where *xxxxx* is the name of the target. Usually only the host assembler is provided with COHERENT.

Assemblers in COHERENT assemble compiler output and (infrequently) assembly language routines. Assemblers exist for the DEC PDP-11, the Zilog Z-8001 and Z8002, the Motorola MC68000, and the Intel iAPX-8086. These assemblers are not necessarily compatible with the manufacturer's assemblers. The *COHERENT Assembler Reference Manual* gives a complete description of the recognized assembly language.

as reads the concatenated source *files* and produces output in **l.out**, or in *ofile* if specified.

The available options include:

–g Classify undefined symbols as *undefined external*.

–l Generate an assembly listing on the standard output.

–o The following argument *ofile* is the object file name. If not specified, **as** places the object in **l.out**.

–x Remove all non-global symbols beginning with **L** from symbol table of object file.

FILES
l.out

SEE ALSO
cc, ld
COHERENT System Manual: **l.out.h**
COHERENT Assembler Reference Manual

DIAGNOSTICS
as reports errors on the standard error stream. It gives a one-letter error code, the line number, the input file name (if more than one specified), and a symbol where appropriate.

a	Addressing error.
b	Byte alignment; for some machines, instructions must begin on even-numbered boundaries.
e	Expression syntax error.
m	Multiple definition of a symbol.
n	Numeric constant syntax error.
p	Phase error.
q	Syntax error in instruction, label or operand.
r	Relocation error.
u	Undefined symbol.

NAME

at – execute commands at given time

USAGE

at [**–v**] [**–c** *command*] *time* [[*day*] **week**] [*file*]
at [**–v**] [**–c** *command*] *time month day* [*file*]

DESCRIPTION

at arranges for commands to be executed at a given time in the future. If *file* is given, **at** takes the commands from it. If the **–c** option is given, the following *command* argument gives the command. If neither is given, **at** reads the standard input for commands.

at interprets a *time* given by a one-digit or two-digit number as an hour specification. It interprets a *time* given by a three-digit or four-digit number as an hour and minutes specification. If the number is followed by a letter **apnm**, **at** assumes AM, PM, Noon or Midnight, respectively; otherwise, it assumes the *time* is given on a 24-hour clock.

In the first form given above, the optional *day* gives a day of the week (lower case, spelled out or first letters). If **week** is specified, **at** moves back the given *time* by one week. In the second form given above, *month* specifies a month name (lower case, spelled out or first letters) and the number *day* specifies a day of the month.

If the **–v** flag is given, **at** prints the time when the commands will be executed.

The current working directory, exported shell variables, file creation mask, user id and group id will be restored when the given command is executed. Output must be redirected appropriately for a command which writes to the standard output.

FILES

/bin/pwd	to find current directory
/usr/lib/atrun	executes scheduled commands
/usr/spool/at	scheduled activity directory
/usr/spool/at/*yymmddhhmm.xx*	commands scheduled at given time

SEE ALSO
COHERENT System Manual: **cron**

NOTES
The daemon **cron** (described in the *COHERENT System Manual*) wakes up **/usr/lib/atrun** periodically to see if scheduled commands should be executed. The frequency with which **cron** executes **atrun** determines the granularity of **at** execution times; it may be changed by altering **/usr/lib/crontab**. **atrun** executes specified commands if it discovers that the given time is past; thus **at** commands will be executed once, even if the system is down at the specified time or if the time is changed.

NAME
awk – report generation, pattern scanning and processing language

USAGE
awk [**–y**] [**–Fc**] [**–f** *progfile*] [*prog*] [*file* ...]

DESCRIPTION
awk is a language for processing input data. Its uses include scanning for patterns, producing reports, and filtering relevant information from input data. It is a general-purpose tool which acts on each input *file* according to an **awk** program. **awk** reads the standard input if no *file* is specified, allowing it to act as a filter in a pipeline. The *file* name '**–**' also means the standard input.

The user can specify the program either as an argument *prog* (usually enclosed in quote marks to prevent interpretation by the shell **sh**) or in the form **–f** *progfile*, where *progfile* contains the **awk** program. If no **–f** option appears, the first non-option argument is the **awk** program *prog*.

The option flag **–y** specifies that any lower-case alphabetic character in a regular expression pattern should match both itself and the corresponding upper-case letter. This is identical to the matching found in the pattern matching program **grep** with the **–y** option.

awk views its input as a sequence of records, each consisting of zero or more fields. By default, newlines separate records and white space (spaces or tabs) separates fields. The option **–F***c* changes the input field separator characters to the characters in the string *c*. An **awk** program can also change the field and record separators. The program can access the values of each field and the entire record through built-in variables.

For details on the construction of **awk** programs, consult the *awk User's Manual*. Briefly, an **awk** program consists of one or more lines, each containing a *pattern* or an *action*, or both. A *pattern* determines whether **awk** performs the associated *action*. It may consist of regular expressions, line ranges, boolean combinations of variables, and beginning and end of input text predicates. If no *pattern* is specified, **awk** executes the *action* (the pattern matches by default).

An *action* is enclosed in braces. The syntax of actions is C-like, and consists of simple and compound statements constructed from constants (numbers, strings), input fields, built-in and user-defined variables, and built-in functions. If an *action* is missing, **awk** prints the entire input record (line).

Unlike **lex** or **yacc**, **awk** does not compile programs into an executable image but rather interprets them directly. Thus **awk** is ideal for quickly-implemented one-shot efforts.

Examples
The following examples illustrate the economy of expression of **awk** programs.

The first example prints all lines containing the string "COHERENT" (identical to **grep COHERENT**):

```
/COHERENT/
```

The built-in variable **NR** is the number of the current input record, so the following program prints the number of records (lines) in the input stream.

```
END     { print NR }
```

The built-in variable **$3** gives the value of the third field of the current record, so the following program sums the elements in column three of an input table and prints the total:

```
        { sum += $3 }
END     { print sum }
```

SEE ALSO
grep, lex, sed, yacc
awk User's Manual

NOTES
There is no way to have a null field, such as is necessary to describe the colon-separated fields in **/etc/passwd**.

awk converts between strings and numbers automatically. Adding 0 to a string forces **awk** to treat it as a number; concatenating "" to a number forces **awk** to treat it as a string.

Maintenance

NAME
bad – maintain bad block list

USAGE
bad *option filesystem* [*block* ...]

DESCRIPTION
The physical device associated with a block special file, typically a hard disk or floppy disk, may have bad blocks which cannot be used reliably because read or write errors occur on them. The COHERENT system includes a mechanism called the *bad block list*, which keeps track of bad blocks so the system can avoid using them.

bad maintains the bad block list for the given *filesystem*, which must be a block special file. Options are available to add blocks to the bad block list, to clear the bad block list, to delete blocks from the bad block list, and to list the blocks on the bad block list.

The *option* argument to **bad** must be exactly one of the characters **acdl**. The **a** option adds each given *block* to the bad block list. The **c** option clears the bad block list. The **d** option deletes each given *block* from the bad block list. The **l** option lists each block on the bad block list.

bad does not deallocate any i-node associated with a block when adding it to the bad block list. **icheck** with the **–s** option should be run immediately after **bad** to correct the problem.

The given *filesystem* should be unmounted if possible. The user who invokes **bad** must have appropriate permissions for the given *filesystem*. For many file systems, only the superuser may use **bad** to change the bad block list.

When the **mkfs** command creates a file system, the prototype specification may include a bad block list for the new file system.

SEE ALSO
icheck, mkfs, umount

NAME
banner – print large sized letters

USAGE
banner [*argument* ...]

DESCRIPTION
banner prints large (nine by eight) sized letters on the standard output. Each *argument* produces one large text output line. If there is no *argument*, each line from the standard input produces one large text output line.

SEE ALSO
lpr, pr

NAME
basename – strip file name

USAGE
basename *file* [*suffix*]

DESCRIPTION
basename strips its argument *file* of any leading directory prefixes. If the result contains the optional *suffix*, **basename** also strips it. **basename** prints the result on the standard output.

NAME

bc – interactive calculator with arbitrary precision

USAGE

bc [–l] [*file* ...]

DESCRIPTION

bc is a language which performs calculations on numbers having arbitrarily many digits. The most common use of **bc** is as an interactive calculator where the user types in arithmetic expressions, in a syntax reminiscent of C. If **bc** is invoked with no *file* arguments on its command line, it reads from the standard input. For example:

Input	Output
bc	
(1000+23)*42	42966
k = 2 ^ 10	
16 * k	16384
2 ^ 100	1267650600228229401496703205376

bc may also be invoked with *file* arguments. After **bc** reads each *file*, it reads the standard input. This provides a convenient way to access programs in files. A library of mathematical functions is available, obtained by using the –l option.

The following is a very brief summary of the facilities provided by bc. More information is available in the *bc Calculator Language Tutorial*.

Comments are enclosed between the delimiters '/*' and '*/'. Names for variables or functions must begin with a lower-case letter and may have any number of subsequent letters or digits; names may not begin with an upper-case letter because of numbers having bases greater than 10. The three built-in variables **obase**, **ibase**, and **scale** represent the number base for printing numbers (default 10), the number base for reading numbers (default 10), and the number of digits after the decimal (radix) point (default 0), respectively. Variables may be simple variables or arrays, and need not be predeclared, with the exception of variables internal to functions. Some examples of variables and array elements are **x25**, **array[10]**, and **number**. Numbers are any string of digits, possibly with one

decimal point. Digits are taken from the ordinary digits (0–9) and then the upper-case letters (A–F), in that order.

Certain names are reserved for use as key words. The key words recognized by **bc** include the following:

if, for, do, while
test conditions and define loops, with syntax identical to C

break, continue
alter control flow within **for** and **while** loops

quit causes **bc** to exit immediately

define *function* (*arg*, ..., *arg*)
defines a **bc** function by a compound statement, as in C

auto *var*, ..., *var*
defines variables that are local to a function, rather than having global scope

return (*value*)
returns a value from a function

scale (*value*)
returns the number of digits to the right of the decimal point in *value*

sqrt (*value*)
returns the square root of *value*

length (*value*)
returns the number of decimal digits in *value*

The following operators are recognized:

+	–	*	/	%	^	++	– –
=	+=	–=	*=	/=	%=	^=	
==	!=	<	<=	>	>=		

The above operators are similar to those in C, with the exception of ^ and ^=, which are exponentiation operators. Expressions may be grouped with parentheses. Statements are separated with semicolons or newlines, and may be made into compound statements with braces. **bc** prints the value of any statement that is an expression but is not an assignment.

As in the editor **ed**, an '!' at the beginning of a line causes that line to be sent as a command to the COHERENT shell **sh**.

The built-in mathematics library contains the following functions and variables:

atan(*z*)	arctangent of *z*
cos(*z*)	cosine of *z*
exp(*z*)	exponential function of *z*
j(*n*,*z*)	*n*th order Bessel function of *z*
ln(*z*)	natural logarithm of *z*
pi	the value of pi to 100 digits
sin(*z*)	sine of *z*

The function below calculates the factorial of its positive integer argument by recursion.

```
/*
 * Factorial function implemented by recursion.
 */
define fact(n) {
        if (n <= 1) return (n);
        return (n * fact(n-1));
}
```

The function below also calculates the factorial of its positive integer argument, this time by iteration.

```
/*
 * Factorial function implemented by iteration.
 */
define fact(n) {
        auto result;

        result = 1;
        for (i=1; i<=n; i++) result *= i;
        return (result);
}
```

FILES
/usr/lib/lib.b source code for the library

SEE ALSO
dc
COHERENT System Manual: **mp**
bc Calculator Language Tutorial

NOTES
Line numbers do not accompany error messages in source files.

NAME
break – exit from shell construct

USAGE
break [*n*]

DESCRIPTION
With no argument, **break** forces an exit from the innermost current **for**, **until** or **while** shell construct. If an argument is given, **break** exits from *n* levels of **for**, **until** or **while** loops.

The shell executes **break** directly.

SEE ALSO
continue, for, sh, until, while

NAME
c – multi-column output

USAGE
c [–w*N*] [–012]

DESCRIPTION
c reads lines from the standard input and writes them in columns on the standard output. The longest input line and the width of the page determine how many columns will fit across the page. The following options apply:

–w*N* Set the width of the page to *N* characters; default is 80.

–0 Multi-column mode 0. Order the fields horizontally across the page.

–1 Multi-column mode 1 (default mode). Order the fields vertically down each column; the last column may be short.

–2 Multi-column mode 2. Order the fields similarly to mode 1, but place blank fields in the last output line rather than the last column.

Options may also be given in the environmental parameter **C**, separated by white space. Command line options override those in the environment. For example,

```
C="-w72 -2"
export C
c -w80 <file1
```

has the same effect as

```
c -w72 -2 -w80 <file1
```

SEE ALSO
pr

NOTES
Too much input may cause **c** to run out of memory.

NAME
cal – print a calendar

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

DESCRIPTION
cal generates a calendar. If neither *year* nor *month* is specified, a calendar of the current month is printed. The *year* must be between 1 and 9999. The *month* may be either the month name (lower case, spelled out or first three letters) or a number between 1 and 12.

For example, try:

```
cal september 1752
```

NOTES
There is no provision for non-English countries, which adopted the Gregorian calendar on a date other than September 3, 1752.

NAME
case – execute commands conditionally according to pattern

USAGE
case *token* in
[*pattern* [| *pattern*] ...) *sequence* ;;] ...
esac

DESCRIPTION
The shell case construct executes commands conditionally according to a pattern. It tests the given *token* successively against each *pattern*, in the specified order. It executes the commands in the *sequence* corresponding to the first matching pattern. Optional '|' clauses specify additional patterns corresponding to a single *sequence*. If no *pattern* matches the *token*, the case construct executes no commands.

As in the shell sh, each *pattern* may include regular characters (which match themselves), special characters '?' (which matches any character except newline) and '*' (which matches any sequence of non-newline characters), and character classes enclosed in brackets '[ ]'; ranges of characters within a class may be separated by '–'. In particular, the last *pattern* in a case construct is often '*', which will match any *token*.

The shell executes case directly.

SEE ALSO
sh

NAME
cat – concatenate/print files

USAGE
cat [–u] [*file* ...]

DESCRIPTION
cat copies each of its *file* arguments to the standard output. A *file* specified by '–' indicates the standard input. If no *file* is specified, cat reads the standard input.

The –u option makes the output unbuffered. Otherwise, cat buffers the output in units of the machine's disk block size (e.g. 512 bytes).

NOTES
Redirecting the output to one of the input files is an error, as the command will never terminate. For example:

```
cat * >out
```


NAME
cc – C compiler

USAGE
cc [*options*] *file* ...

DESCRIPTION
The COHERENT C compiler compiles programs written in the C programming language, as described in *The C Programming Language* by Kernighan and Ritchie. It also supports several advanced features, including the types **void** and **enum**, structure assignment, functions taking structure arguments, and functions returning structure values.

cc is an interface to the C compiler. It takes a list of *options* and *files* from the command line and performs the required negotiation with the C compiler phases, the assembler **as**, and the linker **ld** to compile, to assemble and (optionally) to load complete user programs with the appropriate libraries.

cc assumes each *file* argument ending in **.c** or **.h** to be a C program, and processes it with the C compiler. It assumes each *file* ending in **.s** to be an assembly language program, and processes it with the assembler **as**. It passes any other *file* through to the linker **ld** unchanged.

The normal operation of the cc command is as follows. First, perform the compilations and assemblies; construct the names of the resulting object files by replacing the **.c** or **.s** suffix with a **.o** suffix. Second, link the object files with the C runtime startoff routine, libraries specified by –l*name* options, and the default C library, and leave the result in file **l.out**. Finally, delete the object files. This behavior may be altered by means of command line *options*.

The following *options* are available. cc passes all other *options* through to the linker **ld** unchanged. The scanning is somewhat clever and correctly interprets the **–c**, **–o**, and **–u** options to **ld**. In the **–N**, **–q**, and **–t** options, the letters **p01ab2** refer to phases of the C compiler, **s** to the assembler, **d** to the linker, **l** to libraries, **r** to runtime startup, and **t** to temporary files.

–B[*string*] (Backup) Use alternate versions of the compiler for **cc0**, **cc1**, **cc1a**, **cc1b**, and **cc2**. If a *string* is supplied, cc

prepends it to the names of the phases of the compiler to form the pathnames where these are found. Otherwise, cc prepends the name of the current directory. If a previous −t option was given, only the specified parts of the compiler are affected. Any number of −B and −t options are allowed, with each −t specifying passes affected by the following −B.

−c (Compile) Suppress the link of the resulting object code and prevent the removal of the object files.

−D*name*[=*value*]
(Define) Define *name* to the preprocessor, as if defined by a **#define** directive. If *value* is present, it is used to initialize the definition.

−E (Expand) Run the C preprocessor and write its output to the standard output.

−f (Floating point) Modify the linker command line to include the floating point output routines for **printf**. Normally these routines are replaced with small, fake routines to save space. If this option is not used and a program attempts to print a floating point number (by using the e, f, or g format specifications), the program will print the message "No floating point" and exit.

−I*name* (Include) Specify a directory where the preprocessor should search for files given in **#include** directives. The directory in which the source file is located is always searched first, followed by directories specified in −I options and default system include directories.

−K (Keep) Save the intermediate files of the compilation in the current directory in files derived from *name*.c by replacing .c with a descriptive suffix. The output file from cpp ends with .i to preserve the results of the now defunct −P option.

−l*name* (Library) Pass a library name to the linker. cc expands −l*name* into /lib/lib*name*.a by default. If an alternate library prefix has been specified by −tl and −B*string* options, then −l*name* will expand to *string*lib*name*.a.

−M*string* (Machine) Use alternate versions of cc0, cc1, cc1a, cc1b, cc2, as, /lib*.a, and crts0.o formed by infixing *string* between the directory and pass or file name. The utility of this option is doubtful.

−N[p01ab2sdlrt]*string*
(Name) Rename a specified pass to *string*. For instance, the −p option does an implicit −Nrmcrts0.o to change the runtime startup module.

−O (Optimize) Run the code generated by the C compiler through the peephole optimizer. The optimizer pass is mandatory for the i8086 and m68000 compilers and need not be requested with an −O from those compilers. The optimizer pass is optional for the pdp11 and z8002 compilers; use of the optimizer is recommended on all files except those consisting entirely of initialized tables of data.

−p (Profile) Generate code that counts the number of times a C function is called. If the link edit is not suppressed, replace the standard C runtime startoff routine with a special version that calls the **monitor** subroutine to write a **mon.out** file (suitable for input to the **prof** command).

−q[p01ab2s]
(Quit) Terminate compilation after the specified pass has run.

−S (Suppress) Suppress the assembly and link edit phases. This option is used to obtain an assembly language version of a C program for examination, for example if a compiler problem is suspected. The assembly language output is placed in a file having the same name as the source file, but with the .c suffix replaced by a .s suffix.

−t[p01ab2sdlrt]
(Take) Identify compiler phase(s) and file(s) which should be found in either the current directory or with the prefix string specified in a subsequent −B option. A −t with no argument string affects all compiler phases except the preprocessor.

option is completely different in meaning. The now defunct **-P** option can be emulated with **-Kqp** (keep and quit after cpp).

NAME
cd – change directory

USAGE
cd [*directory*]

DESCRIPTION
The COHERENT command interpreter **sh**, called the shell, keeps track of a directory called the *current working directory*. If a file name is not specified by a complete pathname beginning with '/', the shell prefixes it with the name of the current working directory.

cd changes the current working directory to the specified *directory*. If no *directory* is specified, the **$HOME** directory becomes the current working directory. The shell executes **cd** directly.

FILES
$HOME default working directory name

SEE ALSO
pwd, sh

NAME
check – check file system

USAGE
check [–s] *filesystem* ...

DESCRIPTION
check is a convenient way of checking the consistency of a file system with the **icheck** and **dcheck** commands. **check** acts on each argument *filesystem* in turn; it calls first **icheck** and then **dcheck** on each file system to detect problems.

If –s is specified, **check** attempts to repair any errors automatically. The file system should be unmounted if possible; the system should be rebooted immediately (without typing **sync**) if the root device is involved.

SEE ALSO
clri, dcheck, icheck, ncheck, sync, umount
COHERENT System Manual: **init**
COHERENT Administrator's Guide

NOTES
Certain errors, such as duplicated blocks, cannot be fixed automatically. Decisions must be made by a human.

In earlier releases of COHERENT, **check** acted upon a default file system if none was specified.

NAME
chgrp – change the group owner of files

USAGE
chgrp *group file* ...

DESCRIPTION
chgrp changes the group owner of each *file* to *group*. The *group* may be specified by a valid group name or a valid numerical group ID.

Only the superuser may use **chgrp**.

FILES
/etc/group to convert group name to group ID

SEE ALSO
chmod, chown
COHERENT System Manual: **chown**

SEE ALSO
chgrp, chown, ls, umask
COHERENT System Manual: **chmod, stat, umask**

NOTES
The syntax is painful.

NAME
chown – change the owner of files

USAGE
chown *owner file* ...

DESCRIPTION
chown changes the owner of each *file* to *owner*. The *owner* may be specified by valid user name or a valid numerical user id.

Only the superuser may use **chown.**

FILES
/etc/passwd to convert user name to user id

SEE ALSO
chgrp, chmod
COHERENT System Manual: **chown**

Maintenance

NAME
clri – clear i-node

USAGE
/etc/clri *filesystem inumber ...*

DESCRIPTION
clri zeros out each i-node with a given *inumber* on the specified *filesystem*. The *filesystem* is almost always a device special file corresponding to a disk device. The raw device should be used.

The user must have read and write permission on the *filesystem*.

If the *inumber* corresponds to an open file, the clri has a very high probability of being ineffective; the system maintains a copy of all active i-nodes in core, and this copy will eventually be written out to disk, undoing the effects of clri. To counter this problem, unmount the file system immediately after the clri; in the case of the root file system, reboot.

SEE ALSO
check, dcheck, icheck, umount
COHERENT Administrator's Guide

NAME
cmp – compare bytes of two files

USAGE
cmp [–ls] *file1 file2* [*skip1 skip2*]

DESCRIPTION
cmp compares *file1* and *file2* for equality on a character by character basis. If *file1* is '–', the standard input is used.

Normally, cmp notes the first difference and prints the line and character position (relative to any skips). If cmp encounters end of file on one file but not on the other, it prints the message "EOF on file*n*".

If –l is specified, cmp notes each differing byte by printing the byte position and octal values of the bytes of each file.

If –s is specified, cmp prints nothing but returns the exit status.

If the skip counts are present, cmp reads *skip1* bytes on *file1* and *skip2* bytes on *file2* before performing the comparison.

DIAGNOSTICS
The exit status is 0 for identical files, 1 for non-identical files, and 2 for errors such as bad command usage or inaccessible files.

SEE ALSO
comm, diff, uniq

NAME
col – remove reverse and half line motions

USAGE
col [**–bdfx**] [**–p***n*]

DESCRIPTION
col reads from the standard input and writes to the standard output. It removes reverse and half line motions from the output of **nroff** for the benefit of output devices that cannot perform them. It maintains an image of the page in memory and performs these motions virtually so they do not appear on the output.

col understands four escape sequences: **ESC 7** for reverse line feed, **ESC 8** for half reverse line feed, **ESC 9** for half forward line feed, and **ESC B** for a forward line feed. **ESC** (ASCII 033) is removed from the input stream if followed by any other character.

Eight control characters besides **ESC** are interpreted by **col**. Newline, return, space, backspace and tab carry their usual meaning. **VT** (013) is an alternate form of reverse line feed. The characters **SO** (017) and **SI** (016) signal the start and end of text in an alternate character set. **col** remembers the character set for each character and uses **SO** and **SI** to distinguish them on the output. **col** removes all other control characters from the input stream.

Options:

–b The output device cannot backspace. Only the last of a set of characters destined for a given position will appear.

–d Double space the output. This doubles the length of a document but preserves relative vertical spacing. The **–f** option has precedence.

–f The output device can perform half forward line feeds. Full lines appear single spaced with half lines between them. This is the only situation in which half forward line feeds appear in the output of **col** – reverse line motions never appear.

–x Suppress the default conversion of white space to tabs on output.

–p*n* Set the internal page buffer size to *n* full lines (default 128).

If neither **–f** nor **–d** is chosen, **col** moves non-empty half lines to the next lower full line and pushes all later lines down one line. This can distort the appearance of the document.

SEE ALSO
nroff
COHERENT System Manual: **ascii**

NOTES
Backing up past the start of a document or of the page buffer loses characters.

NAME
comm – print common lines

USAGE
comm [**–123**] *file1 file2*

DESCRIPTION
Both *file1* and *file2* should be sorted in ASCII order. **comm** prints lines unique to *file1* in the first column, lines unique to *file2* in the second, and lines common to both in the third. Any or all columns may be suppressed by indicating the column or columns to suppress in the optional flag. The file '–' means standard input.

SEE ALSO
cmp, diff, sort, uniq

NAME
continue – terminate current iteration of shell construct

USAGE
continue [*n*]

DESCRIPTION
With no argument, **continue** terminates the execution of the current iteration of the innermost **for**, **until** or **while** shell construct; that is, it acts like a branch to the enclosing **done**, after which loop execution may continue or terminate. If an argument is given, **continue** terminates the current iteration of the *n*th enclosing **for**, **until** or **while** loop.

The shell executes **continue** directly.

SEE ALSO
break, for, sh, until, while

NAME
conv – numeric base conversion

USAGE
conv [*number*]

DESCRIPTION
conv converts its *number* argument to various bases (hexadecimal, decimal, octal, binary, and ASCII character) and prints the results on the standard output. If no *number* is given, **conv** reads one number per line from the standard input until end of file.

The input *number* may be given in hexadecimal, decimal, octal, binary, or character format, as shown below. Each example represents the decimal number 97.

Base	Representation
hexadecimal	0x61
hexadecimal	#61
decimal	97
octal	0141
binary	$1100001
character	'a'

conv represents an ASCII control character in its output by preceding the character by a caret '^'. For example, it prints **<ctrl-X>** as '^X'.

SEE ALSO
bc, dd, od, units od

DIAGNOSTICS
conv prints "bad digit" if anything is wrong with the input.

NOTES
conv represents the input *number* internally as a **long** integer. If *number* does not fit in a **long**, **conv** silently truncates it.

NAME
cp – copy files

USAGE
cp *oldfile newfile*
cp *file ... directory*

DESCRIPTION
In the first form, **cp** copies the contents of *oldfile* to *newfile*, which is created if necessary. If *newfile* is a directory, **cp** copies *oldfile* to a file of the same name in directory *newfile*.

In the second form, **cp** places a copy of each *file* under the *directory*.

If a target file did not previously exist, it is given the same mode as the source file. If it did exist, its owner and mode remain unchanged.

A file may not be copied to itself.

SEE ALSO
cpdir, ln, mv

NAME
cpdir – copy directory hierarchy

USAGE
cpdir [*option* ...] *dir1 dir2*

DESCRIPTION
cpdir copies source directory hierarchy *dir1* to target hierarchy *dir2*, which is created if necessary. Either hierarchy may straddle device boundaries.

cpdir preserves as much as possible of the source structure. Files under *dir1* go to identically named files under *dir2*. Links between source files are preserved as links between corresponding target files. Preserved source file attributes include mode, subject to the user's file creation mask. If the user is not the superuser **cpdir** cannot preserve the owner, group and sticky bit in the mode, and the invoking user owns all new files; under the superuser it preserves these as well. In addition, the superuser may "copy" special nodes and pipe nodes; **cpdir** copies only the facility, not the contents. It also preserves real major and minor device numbers of special nodes.

If the target file corresponding to a source file exists and is not a directory, **cpdir** unlinks it before copying. Note that this differs from the action of **cp**.

Options:

–d Preserve the last-modified date instead of using the present date.

–e Print error message and continue execution after an error. The default action is to exit on any error.

–r[*n*] Descend no more than *n* levels in the source hierarchy. Contents of *dir1* are at level 1. If missing, *n* defaults to 1.

–*sname* Suppress the copy of file *name*, which should be the pathname of the file relative to *dir1*.

–t Test only, make no changes. Prints a report of all errors (**–e** is implied), all unlinked target files and other useful information, including a summary of all external links into the target hierarchy broken by the (presumed) unlinking actions.

–u Update regular files. Copy the source only if more recent than the target, or if the target does not exist.

–v Verbose. Prints a file by file account of actions, in addition to the information listed under **–t**.

SEE ALSO
cp
COHERENT System Manual: **link, umask, unlink**

cpdir cpdir
COHERENT Command

NAME
crypt – encrypt/decrypt text

USAGE
crypt [*password*]

DESCRIPTION
crypt is a filter implementing a rotor encryption machine. Similar machines were used for secure communications in World War II by both the Axis (e.g., Germany's 'Enigma') and the Allies (the Hagelin C–48 cipher machine). crypt is distinguished from these machines by using only one rotor with a 256 character alphabet and a keying sequence of period 2^32.

crypt reads text from standard input and writes the encrypted text on standard output. The *password* is used to construct the model of the machine and to start the keying sequence. If no *password* is given crypt prompts for a password on the terminal and disables echo while it is being typed in. The *password* may be up to 10 characters in length but must not be empty. Extra characters are ignored. All characters are legal, although it may not be possible to input certain characters from the terminal.

crypt uses the same *password* for both encryption and decryption. For example:

crypt *password* <*file1* >*file2*
crypt *password* <*file2* >*file3*

leaves *file3* identical to *file1*.

Encrypted files produced by ed under the –x option may be read by crypt, and vice versa, since ed actually uses crypt to do the encryption for it.

Security of a cryptosystem depends on several factors. They include:

1) Brute force attempts to break the system should be infeasible. Passwords should be kept to five or more characters; although the construction of the machine model from the password takes a substantial fraction of a second, it is still plausible that encrypted files could be read by a brute force search of a portion of the password space (say all passwords of length less than four).

2) Cryptanalysis of the basic encryption scheme should be very hard. Analysis of rotor machines is understood, but it is difficult and in most cases probably not worth the trouble.

3) Accidental disclosures of passwords must be avoided. crypt erases *password* as soon as it can to avoid the possibility that it could appear in the output of ps.

4) Privileged access to the system must be guarded. Under COHERENT the security of crypt can be no better than the security governing access to the superuser, since the superuser can essentially do anything. This is probably crypt's most vulnerable point.

FILES
/dev/tty for typed passwords

SEE ALSO
ed
David Kahn, *The Code Breakers*, Macmillan, New York, 1967.
Robert Morris, "The Hagelin Cipher Machine (M – 209)", in *Cryptologia*, Vol. 2, no. 2, July 1978.

NAME
date – print/set date and time of day

USAGE
date [**–u**] [[*yymmdd*]*hhmm*[*.ss*

DESCRIPTION
date prints out the time of day and the current date, including the timezone.

If an argument is given, the system's current time and date is changed. In the above description:

yy gives the last two digits of the current year,
mm gives the two digit month number (1 – 12),
dd gives the two digit day of the month (1 – 31),
hh gives the two digit hour of the day (0 – 23),
mm gives the two digit minute of the hour (0 – 59), and
ss gives the seconds (0 – 59).

At least *hh* and *mm* must be specified—the rest is optional. The date may be changed only by the superuser.

For example,

date 8101211044.25

sets the date to January 21, 1981 10:44:25 a.m.

If option **–u** is specified, dates are set and printed in Co – ordinated Universal Time (GMT) rather than in local time.

The library time conversion routines used by **date** look for the environmental variable **TIMEZONE**, which specifies local time zone and daylight saving time information in the format described in **ctime** (*COHERENT System Manual*).

FILES
/usr/adm/wtmp history of date changes

SEE ALSO
COHERENT System Manual: **ctime, ftime, stime**

NAME
db – assembler-level symbolic debugger

USAGE
db [**–cdefort**] [[*mapfile*] *datafile*]

DESCRIPTION
db provides facilities for running object files under trace control (see **ptrace** in the *COHERENT System Manual*) and for dumping and patching files in a variety of formats.

db takes as arguments a *datafile* whose contents are to be perused and a *mapfile* (always an object file) which supplies a symbol table. If both names are given, the options default to **–c**. If only one name is given, it is the *datafile*; in this case the options default to **–o**. If both names are omitted, *mapfile* defaults to **l.out** and *datafile* defaults to **core**. By default **db** accesses *datafile* with write permission (if possible).

Options specify the format of *datafile*.

–c *datafile* is a core file (produced by a user core dump). **db** checks the name of the command invoking the process that produced the core against the name of the *mapfile*, if given. Pure segments are read from the *mapfile*.

–d *datafile* is a system dump. If only one file is mentioned, *mapfile* defaults to **/coherent**.

–e The next argument is an object file; **db** executes it as a child process and passes it the rest of the command line.

–f Map *datafile* as a straight array of bytes.

–o *datafile* is an object file. If *mapfile* is given, it is another object file providing the symbol table.

–r Read-only: access all files with read, even though write may be possible.

–t Perform input and output for **db** via **/dev/tty**. This permits the debugging of processes whose standard input or output have been redirected.

Commands

In addresses, counts and expressions below each operand may be the symbol '.' (representing the current address), a symbol representing a register, any of the symbols **d**, **i**, or **u** (representing locations 0 in data space, instruction space, and the u-area respectively), a global symbol, or an integer constant (default decimal, leading **0** for octal, **0x** for hex). **db** recognizes the register names **r0**, ... **r7**, **sp**, **pc** for the PDP-11, **r0**, ..., **r15**, **pc** for the Z-8001 and Z8002, and **ax**, **ah**, **al**, **bx**, **bh**, **bl**, **cx**, **ch**, **cl**, **dx**, **dh**, **dl**, **si**, **di**, **bp**, **sp**, **pc**, **cs**, **ds**, **es**, **ss** for the i8086. **db** supports the binary operators '+', '−', '*' and '/' and the unary operators '~', '−' and '*' with their customary meanings and precedences. Parentheses '()' may be used for binding. **db** performs all arithmetic in **longs**.

Every symbol implies a segment to which it refers. This segment in turn determines the default format in which **db** displays data at that location. The segment implied by an expression is that of its leftmost operand. The symbols **d**, **i**, and **u** allow specific segments to be implied in the absence of other symbols.

In the display commands described below, a *format* string consists of any number of the following characters.

^	reset display address to '.'
+	increment display address
−	decrement display address
b	byte
c	char; control and non-chars escaped
C	like 'c' except '\0' not displayed
d	decimal
f	float
F	double
i	machine instruction, disassembled
l	long
n	output '\n'
o	octal
p	symbolic address
s	string terminated by '\0', with escapes
S	string terminated by '\0', no escapes
u	unsigned
w	word
x	hex
Y	time (as in i-node etc.)

Any of the format characters **doux** specifying a numeric base may be followed by one of **blw** specifying a datum size, describing a single datum for display. A format item may also be preceded by a count specifying how many times the item is to be applied.

In the display commands described below, '.' is set to *addr*, *addr* defaults to the current display address if omitted, and *count* defaults to 1.

addr Display the contents of address *addr* in the current format for the implied segment.

addr[,*count*]?[*format*]
Display the implied segment *count* times, starting at *addr*, using *format*. *format* defaults to the previous format for the segment (initially **o** for data and u-area, **i** for instructions). Except where otherwise noted above, **db** increments the display address by the size of the datum displayed after each format item.

In the following commands *addr* defaults to the address where execution stopped (unless otherwise specified); *count* and *expr* default to 1. *commands* is an arbitrary string of **db** commands, terminated by newline. A newline may be included by escaping with '\'.

[*addr*] = Print *addr* in octal. *addr* defaults to '.'.

[*addr*[,*count*]] = *value*
Patch the contents starting at *addr* to the given *value*. *addr* defaults to '.'.

? Print verbose version of last error message.

[*addr*]:a Print *addr* symbolically. *addr* defaults to '.'.

[*addr*]:b[*commands*]
Set breakpoint at *addr*; save *commands* to be executed when breakpoint is encountered. *commands* defaults to ".:a\ni+.?i\n:x".

:br[*commands*]
Set breakpoint at return from current routine. Defaults as above.

COHERENT Command

[*addr*]:c Continue execution from *addr*.

[*addr*]:d[r][s]
Delete breakpoint at *addr*. If optional **r** or **s** is specified, delete return or single-step breakpoint. *addr* defaults to '.'.

[*addr*]:e[*commandline*]
Begin traced execution of the object file at *addr* (default: entry point). The *commandline* is parsed and passed to the traced process. Arg 0 must not be separated from the :e if supplied. Quotes, apostrophes and redirection are parsed as in the shell **sh**, but special characters '?*[]' and shell punctuation '(){}|;' are not. For complete shell command line parsing use the **−e** option.

:f Print type of fault which caused core dump or stopped traced process.

:m Display segmentation map.

:p Display breakpoints.

[*expr*]:q If *expr* is nonzero, quit the current level of command input (see :x). *expr* defaults to 1. End of file is equivalent to :q.

:r Display registers.

[*addr*],[*count*]:s[c][*commands*]
Single-step execution starting at *addr*, for *count* steps, executing *commands* at each step. *commands* defaults to "i+.?i". After a single-step command, an empty command line is equivalent to ".,1:s[c]". If the optional **c** is present, **db** turns off single-stepping at a subroutine call and turns it back on upon return.

[*depth*]:t Print a call traceback to *depth* levels. If *depth* is 0 (default), unwind the whole stack.

[*expr*]:x If *expr* is nonzero, read and execute commands from the standard input up to end of file or :q. *expr* defaults to 1.

FILES
core
l.out
/coherent

SEE ALSO
od
COHERENT System Manual: **core, l.out.h, ptrace**

DIAGNOSTICS
On any command line error (usually inaccessible file) **db** prints an appropriate message and exits. On an interactive command error **db** discards the rest of the input line and prints '?'. At this point typing '?' causes **db** to print a more detailed message.

NAME

dc – desk calculator

USAGE

dc [*file*]

DESCRIPTION

dc is an arbitrary precision desk calculator. It simulates a stacking calculator with ancillary registers. Input must be entered in reverse Polish notation. dc maintains the expected number of decimal places during addition, subtraction, and multiplication, but the user must make an explicit request to maintain any places at all during division.

dc reads input from *file* if specified, and then from the standard input. dc accepts an arbitrary number of commands per line; moreover, spaces need not be left between them.

The *scale factor* of a number is the number of places to the right of its decimal point. The *scale factor register* controls decimal places in calculations. The scale factor does not affect addition or subtraction. It affects multiplication only if the sum of the scale factors of the two operands is greater than it. The result of every division command has as many decimal places as it specifies. It affects exponentiation in that multiplication is performed as many times as the integer part of the exponent indicates; any fractional part of the exponent is ignored.

The dc commands and constructions are:

number

Stack the value of *number*. A number is a string of symbols taken from the digits '0'–'9' and the capital letters 'A'–'F' (usual hexadecimal notation), with an optional decimal point. An underscore '_' as a prefix indicates a negative number. The letters retain values 10–15 respectively regardless of the base chosen by the user.

+ – / * % ^

The arithmetic operations: addition(+), subtraction(–), division(/), multiplication(*), remainder(%), and exponentiation(^). dc pops the two top stack elements, performs the desired operation by calling the multiprecision routine

desired (see mp in the *COHERENT System Manual*), and stacks the result.

c — Clear the stack.

d — Duplicate the top of the stack (so that it occupies the top two positions of the stack).

f — Print the contents of the stack and the values of all registers.

i — Remove the top of the stack and use its integer part as the assumed input base (default 10). The new input base must be greater than 1 and less than 17.

I — Stack the current assumed input base.

k — Remove the top of the stack and put it in the internal scale factor register.

K — Put the value of the internal scale register (which the k command sets) on the top of the stack.

l*x* — Load the value of register *x* to the top of the stack. The value of register *x* is unaltered. *x* may be any character.

o — Remove the top of the stack and use its integer part as the assumed output base (default 10). The specified base may be any positive integer.

O — Stack the current assumed output base.

p — Print the top of the stack. The value remains on the stack.

q — Quit the program; control returns to the shell sh.

s*x* — Remove the top of the stack and store it in register *x*. The previous contents of *x* are destroyed. *x* may be any character.

v — Replace the top of the stack by its square root.

x — Remove the top of the stack, interpret it as a string containing a sequence of dc commands, and execute it. (See [...] below.)

X — Replace the top of the stack by its scale factor (i.e. the number of decimal places it has).

z Place the number of occupied levels of the stack on top of the stack.

[...] Place the bracketed character string on top of the stack. The string may be executed subsequently with the **x** command.

<*x* >*x* =*x* !<*x* !>*x* !=*x*
Remove the top two elements of the stack and compare them. If there is no '!' sign before the relation, execute register *x* if the two elements obey the relation. If a '!' sign is present, execute register *x* if the elements do not obey the relation.

! Interpret the rest of the line as a command to the shell **sh**. Control returns to **dc** after command execution terminates.

SEE ALSO
bc
COHERENT System Manual: **mp**

DIAGNOSTICS
"Stack empty"—not enough stack elements to perform as requested
"Out of pushdown"—no more room on the stack
"Nesting depth"—too many nested execution levels
"Out of space"—too many digits demanded
"Out of headers"—too many numbers being stored

NOTES
For most purposes the infix notation of **bc** is more convenient than the Polish notation of **dc**.

The following example program prints the first twenty Fibbonacci numbers:

```
1sa1sb1sc
[lalbdsa+psb1c1+dsc21<y]sy
lyx
```


Maintenance

NAME
dcheck – directory consistency check

USAGE
dcheck [**–s**] [**–i** *inumber* ...] *filesystem* ...

DESCRIPTION
dcheck performs a consistency check on each specified *filesystem*. It scans all the directories in each *filesystem* and keeps counts of all i-nodes referenced. It compares these counts against the link counts maintained in the i-nodes. **dcheck** notes any discrepancies, and notes allocated i-nodes with a 0 link count.

If the **–i** switch is present, **dcheck** compares each *inumber* in the list against those in each directory. It reports matches by printing the i-number, the i-number of the parent directory, and the name of the entry.

The **–s** switch causes **dcheck** to correct the link count of errant i-nodes to the entry count.

Since **dcheck** is two-pass, the file system should be unmounted. If **–s** is used on the root file system, the system should be rebooted immediately (without performing a **sync**). The raw device should be used.

SEE ALSO
check, clri, icheck, ncheck, sync, umount
COHERENT System Manual: **dir.h**
COHERENT Administrator's Guide

DIAGNOSTICS
If the link count is 0 and there are entries, the file system must be mounted and all entries removed immediately. If the link count is nonzero and the entry count is *larger*, the **–s** option must be used to make the counts agree. **dcheck –s** will effectively clear all inodes with zero entry count and zero link count. In all other cases there may be wasted disk space but there is no danger of losing file data.

The bits in the exit status have the following meaning:

0x01 miscellaneous error (e.g. out of space)
0x02 too hard to fix without human intervention
0x04 difference in link count
0x08 some inodes need clearing

NOTES
In earlier releases of COHERENT, **dcheck** acted upon a default file system if none was specified.

NAME
dd – file conversion

USAGE
dd [*option*=*value*] ...

DESCRIPTION
dd copies an input file to an output file, performing requested conversions. Options include case and character set conversions, byte swapping conversion for other machines, and different input and output buffer sizes. **dd** may be used with raw disk files or raw tape files to do efficient copies with large block (record) sizes, since the byte count for read and write requests can be changed with the **bs** option described below.

The following list gives each available *option*. Any numbers which specify block sizes or seek positions may be written in several ways. A number followed by **w**, **b**, or **k** is multiplied by 2 (for words), 512 (for blocks), or 1024 (for kilobytes), respectively, to obtain the size in bytes. A pair of such numbers separated by **x** is multiplied together to produce the size. All buffer sizes default to 512 bytes if not specified.

bs=*n* Set the size of the buffer for both input and output to *n* bytes.

cbs=*n* Set the conversion buffer size to *n* bytes (used only with character set conversions between ASCII and EBCDIC).

conv=*list* Perform conversions specified by the comma-separated *list*, which may include:

- **ascii** Convert EBCDIC to ASCII.
- **ebcdic** Convert ASCII to EBCDIC.
- **ibm** Convert ASCII to EBCDIC, with mapping more suited to IBM printer codes.
- **lcase** Convert upper case to lower.
- **noerror** Continue processing on I/O errors.
- **swab** Swap every pair of bytes before output.
- **sync** Pad all input buffers with 0 bytes to a size of **ibs**.
- **ucase** Convert lower case to upper.

count = *n* Copy a maximum of *n* input records.

files = *n* Copy a maximum of *n* input files (useful for multifile tapes).

ibs = *n* Set the input buffer size to *n* (normally used if input and output blocking sizes are to be different).

if = *file* Open *file* for input; the standard input is used when no **if =** option is given.

obs = *n* Set the output buffer size to *n*.

of = *file* Open *file* for output; the standard output is used when no **of =** option is given.

seek = *n* Seek to position *n* bytes into the output before copying (does not work on stream data such as tapes, communications devices, and pipes).

skip = *n* Read and discard the first *n* input records.

SEE ALSO
cp, tr, conv
COHERENT System Manual: **ascii, tape**

DIAGNOSTICS
The command reports the number of full and partial buffers read and written upon completion.

NOTES
Because of differing interpretations of EBCDIC, especially for certain more exotic graphic characters such as braces and backslash, no one conversion table will be adequate for all applications. The **ebcdic** table is the American Standard of the Business Equipment Manufacturers Association. The **ibm** table seems to be more practical for line printer codes at many IBM installations.

NAME
deroff – remove text formatting control information

USAGE
deroff [–w] [–x] [*file* ...]

DESCRIPTION
deroff removes text formatting control information from each input text *file*, or from the standard input if no *file* is specified. **deroff** considers all lines beginning with '.' or ' ' to be **nroff** commands and deletes them. **deroff** also recognizes some additional control lines. It deletes **eqn** information (between .EQ and .EN lines), **tbl** information (between .TS and .TE lines) and macro definitions. It also deletes embedded **eqn** requests. It expands source file inclusion with .so and **.nx** requests, with the proviso that no input file is read twice. It also deletes some **troff** escape sequences, such as those for font and size change.

When the **–x** flag is present, **deroff** uses some additional knowledge about the **nroff –ms** macro package.

When the **–w** flag is present, **deroff** divides the remaining text into words and prints them to the standard output, one per line. A word comprises a sequence of letters, digits, and apostrophes which commences with a letter. **deroff** strips apostrophes from the output. All other characters between words are not printed. The spelling checking programs **spell** and **typo** use this option.

SEE ALSO
nroff, spell, typo

NAME
df – print free space on file systems

USAGE
df [**–a**] [**–i**] [**–t**] *filesystem* ...

DESCRIPTION
df prints the number of free disk blocks, the total number of disk blocks available, and the percentage of free disk blocks for each specified *filesystem*.

The options are:

–a	Report on all mounted file systems.
–i *filesystem*	Also give the i-node usage on each *filesystem*.
–t *filesystem*	Also give the total number of blocks on the device.

DIAGNOSTICS
df complains if the argument *filesystem* does not have the appropriate file system format.

FILES
/dev/*

SEE ALSO
mount

NOTES
If you give a *filesystem* name along with the **–a** option, **df** will first report on *filesystem*, then, if *filesystem* is mounted, will report on it again. **df** simply looks at several numbers in the superblock of each *filesystem* to determine validity and the amount of free space; it does not scan the free list. This makes **df** fast but can cause misleading results.

In earlier releases of COHERENT, **df** acted upon a default file system if none was specified.

NAME
diff – summarize differences between two files

USAGE
diff [**–bdefhs**] [**–e** *symbol*] *file1 file2*

DESCRIPTION
diff compares two text files, *file1* and *file2*, and indicates the changes necessary to modify *file1* into *file2*. Normally *file1* is considered to be an older version of *file2*, in which case **diff** will indicate the necessary updates to the older file.

Two special cases involve the input file specifications. If either *file1* or *file2* is '–', the standard input is used for that file. If either is a directory, the actual input file used is the file under the directory with the same name as the specified file.

The default output script has lines in the following format:

a1,a2 C b1,b2

The numbers *a1,a2* refer to line ranges in *file1*, while *b1,b2* refer to ranges in *file2*. As in **ed**, the range is abbreviated to a single number if the first number is the same as the second. The commands, denoted by *C*, are chosen from the **ed** commands '**a**', '**c**', and '**d**'.

After each line describing the range of differences, **diff** displays the differing text from each of the two files. Text associated with *file1* is preceded by '< ', while text associated with *file2* is preceded by '> '.

The comparison of input lines for equality is affected by the **–b** option. When enabled, trailing blanks are ignored and more than one blank in an input line is treated as a single blank. Spaces and tabs are both considered to be blanks for this comparison.

Several options produce output suitable for other purposes. The **–e** *symbol* option produces output suitable for the C preprocessor, containing '#ifdef', '#ifndef', '#else', and '#endif' lines. When the *symbol* is defined to the C pre-processor, *file2* will be produced as output of the pre-processor, otherwise *file1* will be produced.

The **–e** option makes as output an **ed** script which will convert *file1* into *file2*. This is useful for maintaining several versions of a

document or program and a set of changes to it, stored as **ed** scripts. The following illustrates the use of **diff** for this purpose:

```
diff -e file1 file2 >script
echo '1,$p' >>script
ed - file1 <script >file2
```

The **−f** option produces a script in the same manner as the **−e** option, but with line numbers taken directly from the two input files. It will work properly only if applied from end to beginning; it is not directly usable by **ed**.

The **−s** flag produces a **sed** script similar to those produced above.

The **−h** option is intended to be used on large files where the differences are minimal. It uses an algorithm that is not limited by file length but which does not necessarily produce the minimal set of differences. The **−d** option selects the **−h** algorithm only for larger files (larger than 25,000 bytes), and uses the normal algorithm for smaller files.

To modify a script converting *file1* to *file2* into one converting *file2* to *file1*, the line ranges should be interchanged and '**a**' and '**d**' commands should be swapped.

FILES
/tmp/diff*
/usr/lib/diffh to execute the **−h** option

SEE ALSO
cmp, comm, diff3, ed, sed, sort, uniq

DIAGNOSTICS
Exit status is 0 when the files are identical, 1 when they are different, 2 if there was some problem (e.g. could not open a file).

NAME
diff3 − summarize differences among three files

USAGE
diff3 [−ex3] *file1 file2 file3*

DESCRIPTION
diff3 is a generalization of **diff** to three files. Each difference encountered is headed by one of the following separators, which categorizes how many of the three input files differ in a given range. The headers are:

==== all of the files are different

====*n* only the *n*th file differs, where *n* may be 1, 2, or 3.

For each set of changes marked as above, the actual change is indicated for each file using a notation similar to commands to **ed**. For each *filen* the following is printed:

n: *l***a** indicates that text is to be appended after line *l* in *filen*.

n: *l*,*m***c** means that the text from line *l* to line *m* is to be changed for *filen*. The original text from *filen* follows this line. If this text is identical for two of the files, only the latter (higher numbered) of the two is printed.

Options are available to print a script of commands to **ed**. With the **−e** option, a script that will make all changes between *file2* and *file3* to *file1* is produced. This script is based upon all changes flagged with **====** or **====3** separators, as described above.

The **−x** option prints only those changes where all three files differ, i.e. those flagged with **====**.

The **−3** option requests only those changes where *file3* differs. For example, the following command sequence produces a script, applies it to *file1*, and sends the result to the standard output.

```
(diff3 -e file1 file2 file3; echo '1,$p') | ed - file1
```

FILES
/tmp/d3*
/usr/lib/diff3

SEE ALSO
diff, ed

DIAGNOSTICS
An exit status of 0 indicates all three files were identical, 1 indicates differences, and 2 indicates some other failure.

NAME
du – list space used on file system

USAGE
du [–a] [–s] [*directory* **...]**

DESCRIPTION
du prints the total number of 512-byte disk blocks used by each named *directory*. If no *directory* is specified, **du** prints the disk usage of the current directory.

The **–a** (all) option causes **du** to print a line for every file and directory in the substructure. Normally it prints a line only for each directory.

The **–s** (summary) option prints only the line for the top level directory.

du understands links; it adds a file with more than one link to it into the total only once.

SEE ALSO
df, find

NOTES
du does not count indirect blocks, so occasionally a directory does not fit on a file system which appears to contain enough room for it.

NAME
dump – file system dump

USAGE
dump [*options*] [*argument* ...]

DESCRIPTION
dump creates a full or incremental dump of each file system *argument*. File system dumps are in a format which permits both mass restoration of all files on the dump and selective restoration of files by name or by i-number. The dump device may be either tape or floppy disk; **dump** prompts the user to mount additional volumes if necessary.

A file system dump includes all files changed since the *dump since* date, plus any directories between the root of the dumped file system and any dumped file (for the benefit of **dumpdir**).

The *options* string specifies both the dump since date and the dump processing options. It is made up of characters from the set **0123456789bdfsSuv**, which have the following meanings.

0–9 The digit gives the level number of the dump. The dump since date is the most recent date in the dump date file **/etc/ddate** associated with this file system and having a level number less than the current dump level. A level 0 dump is always a full dump, since there can never be an entry in the dump date file with a level number less than 0.

b The next *argument* gives the output tape blocking factor. The blocking factor is the number of 'dumpdata' structures in each tape block. The default blocking factor is 20.

d The next *argument* gives the density of the output tape in bytes per inch. The default density is 1600 bpi. **dump** uses the density to compute the quantity of tape used.

f The next *argument* gives the pathname of the output file. If no **f** option is given, **/dev/dump** is assumed.

s The next *argument* gives the length of the dump tape in feet. **dump** keeps a running total of the quantity of tape it has written, and it asks for a new reel if it appears that the end of the reel is near. The default length is 2300 feet.

S The next *argument* gives the size of the dump floppy disk in blocks.

u If the dump completes without error, update the record of successful dumps kept in file **/etc/ddate**. There is an entry in this file for every file system and every dump level.

v Inform the user of the 'dump since' date and the length of tape used in feet. The length is useful for computing the quantity of tape remaining if multiple dumps are written onto a single reel of tape.

If no level number is given, **dump** assumes the *options* **9u**.

FILES

/dev/dump	default dump device
/etc/ddate	dump date file

SEE ALSO
dumpdate, dumpdir, restor
COHERENT Administrator's Guide

DIAGNOSTICS
Most errors are fatal, caused by a table overflowing or a read or write error on the input or output device.

NOTES
Earlier releases of COHERENT included a version of **dump** with a different default dump device and with a default file system to dump. The headers of the old and new dump formats are incompatible.

NAME
dumpdate – print dump dates

USAGE
dumpdate [*filesystem* ...]

DESCRIPTION
dumpdate reads through the dump date file **/etc/ddate** and displays the dump date records associated with each specified *filesystem*.

If no *filesystem* is specified, the records for all file systems are displayed.

FILES
/etc/ddate dump date file

SEE ALSO
dump, dumpdir, restor

NAME
dumpdir – print the directory of a dump

USAGE
dumpdir [**af** [*argument* ...]]

DESCRIPTION
dumpdir reads through a file system dump created by the **dump** command, gathers up its directory blocks, and displays the names and i-numbers of all files on the dump.

The **a** option causes **dumpdir** to display the directory entries for '.' and '..', which are normally suppressed.

The **f** option causes the next *argument* to be taken as the pathname of the dump device, which is otherwise assumed to be **/dev/dump**.

If no options are specified, **dumpdir** reads from the default dump device **/dev/dump** and suppresses the printing of '.' and '..' entries.

FILES

/dev/dump	default dump device
/tmp/dd*XXXXXX*	to hold directory blocks

SEE ALSO
dump

DIAGNOSTICS
If the directory blocks of the dump continue onto additional dump tape reels or floppy disks, **dumpdir** prompts the user to mount the next reel or disk. Respond with a newline when the reel or disk is ready.

NAME
echo – print arguments

USAGE
echo [**–n**] [*argument* ...]

DESCRIPTION
echo prints each *argument* on the standard output. It prints a space between each *argument*. **echo** appends a newline to the end of the output unless the **–n** flag is present.

NAME
ed – interactive text editor

USAGE
ed [–] [**–x**] [**+cmopsv**] [*file*]

DESCRIPTION
ed is a line-oriented interactive text editor. It provides the ability to enter or change text, to locate and replace text patterns, to move or copy blocks of text, and to print parts of the text. **ed** can read text from input files and can write all or part of the edited text to other files.

This section gives a brief description of **ed** commands. The *ed Interactive Editor Tutorial* gives a detailed tutorial on how to use **ed**.

ed reads commands from the standard input, usually one command per line. Normally, **ed** does not prompt for commands. If the optional *file* argument is given, **ed** edits the given file, as if the *file* were read with the **e** command described below.

ed manipulates text in memory rather than on a file. No changes to a file occur until the user writes edited text with the **w** command. Since **ed** works with text in memory, it fails on very large files. Large files can be divided with **split** or edited with the stream editor **sed**.

ed remembers some information to simplify its commands. The *current line* is typically the line most recently edited or printed. When **ed** reads in a file, the last line read becomes the current line. The *current file name* is the last file name specified in an **e** or **f** command. The *current search pattern* is the last pattern specified in a search specification.

Line specifiers
ed identifies text lines by integer line numbers, beginning with 1 for the first line. Several special forms identify a line or a range of lines. These forms are:

n	A decimal number *n* specifies the *n*th line of the text.
.	A period '.' specifies the current line.
$	A dollar sign '$' specifies the last line of the text.
+, –	Simple arithmetic may be performed on line numbers.

/pattern/ Search forward from the current line for the next occurrence of the *pattern*. If **ed** finds no occurrence before the end of the text, the search wraps to the beginning of the text. Patterns, also called regular expressions, are described in detail below.

?pattern? Search backwards from the current line to the previous occurrence of the *pattern*. If **ed** finds no occurrence before the beginning of the text, the search wraps to the end of the text.

'x Lines marked with the **k***x* command described below are identified by *'x*. The *x* may be any lower-case letter.

n,*m* Line specifiers separated by a comma ',' specify the range of lines between the two given lines, inclusive.

n;*m* Line specifiers separated by a semicolon ';' specify the range of lines between the two given lines, inclusive. Normally, **ed** updates the current line after it executes each command. If a semicolon ';' rather than a comma separates two line specifiers, **ed** updates the current line before reading the second.

* An asterisk '*' specifies all lines; equivalent to **1,$**.

Commands

ed commands consist of a single letter, optionally preceded by one or two line specifiers giving the line or lines to which the command applies.

The following command summary uses the notations [*n*] and [*n*[,*m*]] to refer to an optional line specifier and an optional range, respectively. These default to the current line when omitted, except where otherwise noted. A semicolon ';' may separate two line specifiers instead of a comma.

. Print the current line. Also, a line containing only a period '.' marks the end of appended, changed, or inserted text.

[*n*] Print given line. If no line number is given (i.e. the command line consists only of newline), print the line following the current line.

[*n*]= Print the specified line number (default: last line number).

[*n*]& Print a screen of 23 lines; equivalent to *n*,*n* + 22**p**.

!*line* Pass the given *line* to the shell **sh** for execution. **ed** prompts with an exclamation point '!' when execution is completed.

? Print a brief description of the most recent error.

[*n*]**a** Append new text after line *n*. Terminate new text with line containing only a period '.'.

[*n*[,*m*]]**c** Change specified lines to new text. Terminate new text with line containing only a period '.'.

[*n*[,*m*]]**d**[**p**] Delete specified lines. If **p** follows, print new current line.

e [*file*] Edit the specified *file* (default: current file name). An error occurs if there are unsaved changes. Reissuing the command after the error message forces **ed** to edit the *file*.

E [*file*] Edit the specified *file* (default: current file name). No error occurs if there are unsaved changes.

f [*file*] Change the current file name to *file* and print it. If *file* is omitted, print the current file name.

[*n*[,*m*]]**g**/[*pattern*]/*commands*
Globally execute *commands* for each line in the specified range (default: all lines) containing the *pattern* (default: current search pattern). The *commands* may extend over several lines, with all but the last terminated by '\'.

[*n*]**i** Insert text before line *n*. Terminate new text by a line containing only a period '.'.

[*n*[,*m*]]**j**[**p**] Join specified lines into one line. If *m* is not specified, use range *n*,*n* + 1. If no range is specified, join the current line with the next line. With optional **p**, print resulting line.

[*n*]**k***x* Mark given line with lower-case letter *x*.

[*n*[,*m*]]**l** List selected lines, interpreting non-graphic characters.

[*n*[,*m*]]m[*d*]
Move selected lines to follow line *d* (default: current line).

o*options* Change the given *options*. The *options* may consist of an optional sign '+' or '−', followed by one or more of the letters 'cmopsv'. Options are explained below.

[*n*[,*m*]][p]
Print selected lines. The p is optional.

q Quit editing and exit. An error occurs if there are unsaved changes. Reissuing the command after the error message forces ed to exit.

Q Quit editing and exit. No error occurs if there are unsaved changes.

[*n*]r [*file*]
Read *file* into current text following given line (default: last line).

[*n*[,*m*]]s[*k*]/[*pattern1*]/*pattern2*/[g][p]
Search for *pattern1* (default: remembered search pattern) and substitute *pattern2* for *k*th occurrence (default: first) on each line of the given range. If g follows, substitute every occurrence on each line. If p follows, print the resulting current line.

[*n*[,*m*]]t[*d*]
Transfer (copy) selected lines to follow line *d* (default: current line).

[*n*]u[p] Undo effect of last substitute command. If optional p specified, print undone line. The specified line must be the last substituted line.

[*n*[,*m*]]v/[*pattern*]/*commands*
Globally execute *commands* for each line in the specified range (default: all lines) *not* containing the *pattern* (default: current search pattern). The *commands* may extend over several lines, with all but the last terminated by '\'. The v command is like the g command, except the sense of the search is reversed.

[*n*[,*m*]]w [*file*]
Write selected lines (default: all lines) to *file* (default: current file name). The previous contents of *file*, if any, are lost.

[*n*[,*m*]]W [*file*]
Write specified lines (default: all lines) to the end of *file* (default: current file name). Like w, but appends to *file* instead of truncating it.

x Encrypt/decrypt text using crypt. ed prompts for an encryption password and applies the resulting key to encrypt or decrypt on each subsequent e, r, or w command. An empty password turns off encryption.

Patterns

Substitution commands and search specifications may include *patterns*, also called *regular expressions*. A non-special character in a pattern matches itself. Special characters include the following.

^ Matches beginning of line, unless it appears immediately after '[' (see below).

$ Matches end of line.

* Matches zero or more repetitions of preceding character.

. Matches any character except newline.

[*chars*] Matches any one of the enclosed *chars*. Ranges of letters or digits may be indicated using '−'.

[^*chars*]
Matches any character *except* one of the enclosed *chars*. Ranges of letters or digits may be indicated using '−'.

c Disregard special meaning of character *c*.

\(*pattern*\)
Delimit substring *pattern* for use with *d*, described below.

The replacement part *pattern2* of the substitute command may also use the following:

& Insert characters matched by *pattern1*.

\d Insert substring delimited by *d*th occurrence of delimiters '\(' and '\)', where *d* is a digit.

Options
The user may specify **ed** options on the command line, in the environment, or with the **o** command. The available options are:

c Print character counts on **e**, **r**, and **w** commands.

m Allow multiple commands per line.

o Print line counts instead of character counts on **e**, **r**, and **w** commands.

p Prompt with '*' for each command.

s Match lower-case letters in a *pattern* to both upper-case and lower-case text characters.

v Print verbose versions of error messages.

The **c** option is normally set, and all others are normally reset. Options may be set on the command line with a leading '+' sign. The '−' command line option resets the **c** option. The '−**x**' command line option causes **ed** to encrypt and decrypt text written to and read from files, as with the **x** command.

Options may be set in the environment with an assignment, such as

```
ED=+cv
```

Options may be set with the '+' prefix or reset with the '−' prefix.

FILES

/bin/crypt	to perform encryption
/tmp/ed*XXXXXX*	temporary file

SEE ALSO
crypt, **sed**, **sh**, **split**
ed Interactive Editor Tutorial

DIAGNOSTICS
ed usually prints only the diagnostic '?' on any error. When the verbose option **v** is specified, the '?' is followed by a brief description of the nature of the error.

NAME
egrep − extended pattern search

USAGE
egrep [*option* ...] [*pattern*] [*file* ...]

DESCRIPTION
egrep is an extended and faster version of **grep**. It searches each *file* for occurrences of *pattern* (also called a regular expression). If no *file* is specified, it searches the standard input. Normally, it prints each line matching the *pattern*.

The *patterns* accepted by **grep** are the same as those used by **ed**. However, **egrep** accepts an extended set of patterns. The additional special characters in **egrep** patterns are:

| Matches the preceding pattern *or* the following pattern. For example, the pattern **cat|dog** matches either **cat** or **dog**. A newline within the *pattern* has the same meaning as '|'.

\+ Matches one or more occurrences of the immediately preceding pattern element; like '*', except it matches at least one occurence instead of zero or more occurrences.

? Matches zero or one occurrence of the preceding pattern element.

(...) Parentheses may be used to group patterns. For example, **(Alex)+** matches a sequence of one or more occurrences of the four letters **Alex**.

Since some special characters used in the *pattern* are also special to the shell **sh**, patterns containing those characters must be quoted by enclosing the *pattern* within single quotes.

The available *options* are:

−b With each output line, print the block number in which the line started (used to search file systems).

−c Print the count of matching lines rather than the lines.

−e The next argument is *pattern* (useful if the pattern starts with '−').

−f The next argument is a file containing a list of patterns separated by newlines; there is no *pattern* argument.

-h When more than one *file* is specified, output lines are normally accompanied by the file name; **-h** suppresses this.

-l Print the name of each file containing matching lines rather than the lines.

-n The line number in the file accompanies each line printed.

-s Suppress all output, just return status.

-v Print a line only if the pattern is *not* found in the line.

-y Lower-case letters in the pattern match lower-case *and* upper-case letters on the input lines. A letter escaped with '\' in the pattern must be matched in exactly that case.

SEE ALSO
awk, ed, expr, grep, lex, sed

DIAGNOSTICS
egrep returns an exit status of 0 for success, 1 for no matches, 2 for error.

NOTES
Besides the difference in the range of patterns allowed, **egrep** is much faster than **grep**, frequently by as much as an order of magnitude. This is because it uses a deterministic finite automaton (DFA) for the search instead of grep's non-deterministic finite automaton (NFA). **egrep** builds the DFA dynamically, so it begins doing useful work immediately. As a result, **egrep** is faster than **grep** on almost any length file.

NAME
enroll – enroll user in public key cryptosystem

USAGE
enroll

DESCRIPTION
To facilitate private communication between users, the COHERENT system includes a public key cryptosystem based on a trapdoor knapsack algorithm. Each user has two keys, a *public* encrypting key and a *private* decrypting key. Anyone can encrypt a message to a user, using the user's public key. Encrypted text can be read only by using the corresponding private key. The public key cryptosystem allows a user to send private messages to other users, even though all information in the COHERENT system is accessible to certain users (such as the superuser **root**).

Cryptosystem users do not deal directly with keys, but rather with a *passphrase* which generates keys. A passphrase contains up to 128 characters, terminated by newline; all ASCII characters are significant. The passphrase must always be entered from a terminal and echoing is disabled while it is typed in.

enroll enrolls a user in the cryptosystem. It asks for a passphrase, generates a public key with it, and stores the key in the public key file **/usr/spool/pubkey/***user*. A new passphrase must be repeated to avoid typographical errors.

If the user is already enrolled, **enroll** allows generation of a new public key. It asks for the current passphrase; if it agrees with the existing public key, **enroll** asks for a new passphrase and makes the change. **enroll** removes the user's public key file if the new passphrase is empty.

xencode encodes a private message. **xdecode** decodes a private message. **xmail** sends or reads mail through the cryptosystem.

FILES
/usr/spool/pubkey/*user* for *user*'s public key

SEE ALSO
crypt, xdecode, xencode, xmail
Merkle, Ralph Charles, *Secrecy, Authentication, and Public Key Systems,* University Microfilms International, Ann Arbor, Michigan, 1979.

NOTES
Encoding and decoding messages can be slow, as it requires extensive calculations with multiple-precision integers.

No warranty is made as to the security of the knapsack encryption system. Adi Shamir of the Weizmann Institute in Israel has recently proven that a somewhat simpler trapdoor knapsack encryption is solvable in polynomial time. Future advances may compromise this system as well.

NAME
eval – evaluate arguments

USAGE
eval [*token* ...]

DESCRIPTION
The shell **sh** normally evaluates each token of an input line before executing it. During evaluation, the shell performs parameter, command, and file name pattern substitution, as described in **sh**. The shell does *not* interpret special characters after performing substitution.

eval is useful when an additional level of evaluation is required. **eval** evaluates its arguments and treats the result as shell input. For example,

```
A='>file'
echo a b c $A
```

simply prints the output

```
a b c >file
```

because '>' has no special meaning after substitution, but

```
A='>file'
eval echo a b c $A
```

redirects the output

```
a b c
```

to file. Similarly,

```
A='$B'
B='string'
echo $A
eval echo $A
```

prints

```
$B
string
```

In the first **echo** the shell performs substitution only once.

The shell executes **eval** directly.

SEE ALSO
sh

NAME
exec – execute command directly

USAGE
exec [*command*]

DESCRIPTION
The shell **sh** normally executes commands with a **fork** system call, which creates a new process. The shell command **exec** directly executes the given *command* with an **exec** system call instead. Normally, this terminates execution of the current shell.

If the *command* consists only of redirection specifications, as described in **sh**, **exec** redirects the input or output of the current shell accordingly without terminating it. If the *command* is omitted, **exec** has no effect.

SEE ALSO
sh
COHERENT System Manual: **exec, fork**

NAME
exit – exit from noninteractive shell

USAGE
exit [*status*]

DESCRIPTION
exit terminates a noninteractive shell **sh**. If the optional *status* is specified, the shell returns it; otherwise, the previous status is unchanged. From an interactive shell, **exit** sets the *status* if specified, but does not terminate the shell.

The shell executes **exit** directly.

SEE ALSO
sh

NAME
export – add shell variables to environment

USAGE
export [*name* ...]

DESCRIPTION
When the shell **sh** executes a command, it passes the command an *environment*. By convention, the environment consists of assignments, each of the form *name=value*. A command may look for information in the environment or may simply ignore it.

The shell places the *name* and the value of each shell variable which appears in an **export** command into the environment of subsequently executed commands. It does not place a shell variable into the environment until it appears in an **export** command.

With no arguments, **export** prints the name and the value of each shell variable currently marked for export.

The shell executes **export** directly.

SEE ALSO
sh
COHERENT System Manual: **environ, exec**

NAME

expr – compute a command line expression

USAGE

expr *argument ...*

DESCRIPTION

The arguments to **expr** form an expression. **expr** evaluates the expression and writes the result on the standard output. Among other uses, **expr** lets the user perform arithmetic in shell command files.

Each *argument* is a separate token in the expression. An argument has a logical value 'false' if it is a null string or has numerical value zero, 'true' otherwise. Integer arguments consist of an optional sign followed by a string of decimal digits. The range of valid integers is that of signed long integers. No check is made for overflow or illegal arithmetic operations. Floating point numbers are not supported.

The following list gives each **expr** operator and its meaning. The list is in order of increasing operator precedence; operators of the same precedence are grouped together. All operators associate left to right except the unary operators '!', '–' and '**len**', which associate right to left. The spaces shown are significant—they separate the tokens of the expression.

{ *expr1* , *expr2* , *expr3* }
: returns *expr2* if *expr1* is logically true, and *expr3* otherwise. Alternatively, { *expr1* , *expr2* } is equivalent to { *expr1* , *expr2* , **0** }.

expr1 | *expr2*
: returns *expr1* if it is true, *expr2* otherwise.

expr1 & *expr2*
: returns *expr1* if both are true, 0 otherwise.

expr1 relation expr2
: where *relation* is one of <, <=, >, >=, ==, or !=, returns 1 if the *relation* is true, 0 otherwise. The comparison is numeric if both arguments can be interpreted as numbers, lexicographic otherwise. The lexicographic comparison is the same as **strcmp** (see **string**, in the *COHERENT System Manual*).

expr1 + *expr2*
expr1 – *expr2*
: adds or subtracts the integer arguments. The expression is invalid if either *expr* is not a number.

expr1 * *expr2*
expr1 / *expr2*
expr1 % *expr2*
: multiplies, divides or takes remainder of the arguments. The expression is invalid if either *expr* is not numeric.

expr1 : *expr2*
: matches patterns (regular expressions). *expr2* specifies a pattern in the syntax used by **ed**. It is compared to *expr1*, which may be any string. If the \(...\) pattern occurs in the regular expression the matching operator returns the matched field from the string; if there is more than one \(...\) pattern the extracted fields are concatenated in the result. Otherwise, the matching operator returns the number of characters matched.

len *expr*
: returns the length of *expr*. It behaves like **strlen** (see **string**, in the *COHERENT System Manual*). Note that *len* is a reserved word in **expr**.

! *expr*
: logical negation returns 0 if *expr* is true, 1 otherwise.

– *expr*
: unary minus returns the negative of its integer argument; if the argument is non-numeric the expression is invalid.

(*expr*)
: returns the *expr*. The parentheses allow grouping expressions in any desired way.

Several operators have special meanings to the shell **sh**, and must be quoted to be interpreted correctly. The characters that must be quoted are { } ( ) < > & | *.

COHERENT Command

SEE ALSO
ed, sh, test
COHERENT System Manual: **string**

DIAGNOSTICS
expr returns 0 if the expression is true, 1 if false, and 2 if an error occurs. In the latter case an error message is also printed.

NAME
false – unconditional failure

USAGE
false

DESCRIPTION
false does nothing, unsuccessfully. It can be useful in shell scripts:

```
${CMD-false}
exit
```

DIAGNOSTICS
Exit status is 1.

NAME
file – guess type of a file

USAGE
file *name* ...

DESCRIPTION
file attempts to guess the type of each argument *name* specified. It examines regular files (i.e. not directories or special files) to make an educated guess about their format.

file recognizes the following classes of text files: files of commands to the shell; files containing the source for a C program; files containing **yacc** or **lex** source; files containing assembly language source; files containing unformatted documents which may be passed to **nroff**; and plain text files, which fit into none of the above categories.

file recognizes the following classes of non-text or binary data files: the various forms of archives, object files and load modules for various machines, and other binary data files.

SEE ALSO
ls, sh, size

NOTES
Since **file** only reads a finite amount of data to determine the class of a text file, mistakes can happen.

NAME
find – search for files satisfying a pattern

USAGE
find *directory* ... [*expression* ...]

DESCRIPTION
find traverses each given *directory*, testing each file or subdirectory found with the *expression* part of the command line. The test can be the basis for deciding whether to process the file with a given command.

If the command line specifies no *expression* or specifies no execution or printing (**–print**, **–exec**, or **–ok**), by default **find** prints the pathnames of the files found.

In the following, *file* means any file: directory, special file, ordinary file, and so on. Numbers represented by *n* may be optionally prefixed by a '+' or '–' sign to signify values greater than *n* or less than *n*, respectively.

find recognizes the following *expression* primitives:

–atime *n*	Matches if the file was accessed in the last *n* days.
–ctime *n*	Matches if the i-node associated with the file was changed in the last *n* days, as by **chmod** (*COHERENT System Manual*).
–exec *command*	Matches if *command* executes successfully (has a zero exit status). The *command* consists of the following arguments to **find**, terminated by a semicolon ';' (escaped to get past the shell). **find** substitutes the current pathname being tested for any argument of the form '{ }'.
–group *name*	Matches if the file is owned by group *name*. If *name* is a number, the owner must have that group number.
–inum *n*	Matches if the file is associated with i-number *n*.
–links *n*	Matches if the number of links to the file is *n*.
–mtime *n*	Matches if the most recent modification to the file was *n* days ago.

–name *pattern*	Matches if the file name corresponds to *pattern*, which may include the special characters '*', '?', and '[...]' recognized by the shell **sh**. The *pattern* matches only the part of the file name after any slash ('/') characters.
–newer *file*	Matches if the file is newer than *file*.
–nop	Always matches; does nothing.
–ok *command*	Same as **–exec** above, except prompts interactively and only executes *command* if the user types response 'y'.
–perm *octal*	Matches if owner, group, and other permissions of the file are the *octal* bit pattern, as described in **chmod**. When *octal* begins with a '–' character, more of the permission bits (setuid, setgid, and sticky bit) become significant.
–print	Always matches; prints the file name.
–size *n*	Matches if the file is *n* (512-byte) blocks in length.
–type *c*	Matches if the type of the file is *c*, chosen from the set **bcdfmp** (for block special, character special, directory, ordinary file, multiplexed file, or pipe, respectively).
–user *name*	Matches if the file is owned by user *name*. If *name* is a number, the owner must have that user number.
exp1 exp2	Matches if both expressions match. **find** evaluates *exp2* only if *exp1* matches.
exp1 **–a** *exp2*	Matches if both expressions match, as above.
exp1 **–o** *exp2*	Matches if either expression matches. **find** evaluates **exp2** only if **exp1** does not match.
! *exp*	Matches if the expression does *not* match.
(*exp* **)**	Parentheses are available for expression grouping.

Examples
A **find** command to print the names of all files and directories in user **fred**'s directory is:

```
find /usr/fred
```

The following more complicated **find** command prints out information on all **core** and object (**.o**) files that have not been changed for a day. Because some characters are special both to **find** and **sh**, they must be escaped with '\' to avoid interpretation by the shell.

```
find / \( -name core -o -name \*.o \) -mtime +1 \
-exec ls -l { } \;
```

SEE ALSO
ls, sh, test
COHERENT System Manual: **chmod**

NAME
for – execute commands for tokens in list

USAGE
for *name* [**in** *token* ...]
do
sequence
done

DESCRIPTION
The shell **for** loop construct successively assigns the shell variable *name* each value in the *token* list and then executes the commands in the given *sequence*. If the **in** clause is omitted, **for** successively assigns *name* the value of each positional parameter to the current script ('$@'). Because the shell recognizes a reserved word only as the unquoted first word of a command, both **do** and **done** must occur unquoted at the start of a line or preceded by ';'.

The shell commands **break** and **continue** may be used to alter control flow within a **for** loop.

The shell **sh** executes **for** directly.

SEE ALSO
break, continue, sh

Games

NAME
fortune – generate a fortune cookie

USAGE
/usr/games/fortune [*file*]

DESCRIPTION
fortune generates a message reminiscent of the contents of a fortune cookie. If the *file* argument is specified, **fortune** takes the fortune from it instead of from the default **/usr/games/lib/fortunes**.

FILES
/usr/games/lib/fortunes default fortunes

NAME
from – generate list of numbers

USAGE
from *start* **to** *stop* [**by** *incr*]

DESCRIPTION
from prints a list of integers on the standard output, one per line. It prints *start* and then prints successive numbers generated by adding *incr* (default: 1) to the previous number. It continues until the generated value passes *stop*. Each of *start*, *stop* and optional *incr* is a decimal integer with an optional leading '–' sign.

Typical uses of **from** include generating a file of numbers and generating a loop index for the shell **sh**. The following example creates special files for eight terminal ports.

```
for i in `from 0 to 7`
do
        /etc/mknod /dev/tty3$i c 3 $i
done
```

DIAGNOSTICS
from prints an error message if the generated list is empty.

NAME
grep – pattern search

USAGE
grep [*option* ...] [*pattern*] [*file* ...]

DESCRIPTION
grep searches each *file* for occurrences of the *pattern* (sometimes called a regular expression). If no *file* is specified, **grep** searches the standard input. The *pattern* is given in the same manner as for **ed**. Normally **grep** prints each line matching the *pattern*.

The following options are available.

–b	With each output line, print the block number in which the line started (used to search file systems).
–c	Print the count of matching lines rather than the lines.
–e	The next argument is *pattern* (useful if the pattern starts with '–').
–f	The next argument is a file containing a list of patterns separated by newlines; there is no *pattern* argument.
–h	When more than one *file* is specified, output lines are normally accompanied by the file name; **–h** suppresses this.
–l	Print the name of each file containing matching lines rather than the lines.
–n	The line number in the file accompanies each line printed.
–s	Suppress all output, just return status.
–v	Print a line if the pattern is *not* found in the line.
–x	Print the line only if it is exactly the same as the pattern; treat wildcards in the pattern as plain text.
–y	Lower-case letters in the pattern match lower-case *and* upper-case letters on the input lines.

SEE ALSO
awk, ed, egrep, expr, lex, sed

DIAGNOSTICS
grep returns an exit status of 0 for success, 1 for no matches, 2 for error.

NOTES
egrep is an extended and faster version of **grep**.

NAME
help – print concise command description

USAGE
help [*command* ...]

DESCRIPTION
help prints a concise description of the options available for each specifed *command*. If the *command* is omitted, **help** prints a simple description of itself, followed by information about the command given by **$LASTERROR**, which is the last command returning a nonzero exit status.

The information provided by **help** is intermediate between the usage message printed by a command when invoked erroneously and the detailed description given by the **man** command. The primary purpose of **help** is to refresh the memory of a user who has forgotten a *command* option.

help prints information normally found between **.HS** and **.HE** directives in the **nroff** source for the manual pages. **nroff** ignores this information. If **help** finds no information in the manual pages, it looks in **/etc/helpfile** for additional system information and in **$HELP** for user-specific information. Information about a given *command* begins with a line

#command

and ends with the next line beginning with '**#**' in **/etc/helpfile** or **$HELP**. **help** constructs the index file **/etc/helpindex** to make subsequent searches of **/etc/helpfile** faster.

FILES

/etc/helpfile	additional system information
/etc/helpindex	index for helpfile
/usr/man/cmd/*	to extract summaries
$HELP	user information
$LASTERROR	default command name

SEE ALSO
learn, man
COHERENT System Manual: **man**

NOTES
In earlier releases of COHERENT, **help** did not use **/etc/helpindex** and did not print **LASTERROR** information.

NAME
icheck – i-node consistency check

USAGE
icheck [**–s**] [**–b** *N* ...] [**–v**] *filesystem* ...

DESCRIPTION
Each block in a file system must be either free (i.e. in the free list) or allocated (i.e. associated with exactly one i-node). **icheck** examines each specified *filesystem*, printing block numbers which are claimed by more than one i-node, or claimed by both an i-node and the free list. It also checks for blocks which appear more than once in the block list of an i-node or in the free list.

The option **–v** (verbose) causes **icheck** to print a summary of block usage in the *filesystem*. The option **–s** causes **icheck** to ignore the free list, to note which blocks are claimed by i-nodes, and to rebuild the free list with the remainder. A list of block numbers may be submitted with the **–b** flag; **icheck** prints the data structure associated with each block as the file system is scanned.

The raw device should be used, and the *filesystem* should be unmounted if possible. If this is not possible (e.g. on the root file system) and the **–s** option is used, the system must be rebooted immediately to expunge the obsolete superblock.

The exit status bits for a bad return are:

0x01	miscellaneous error (e.g. out of space)
0x02	too hard to fix without human intervention
0x04	bad free block
0x08	missing blocks
0x10	duplicates in free list
0x20	bad block in free list

SEE ALSO
check, clri, dcheck, ncheck, sync, umount
COHERENT Administrator's Guide

DIAGNOSTICS
"dups in free" indicates a block is in the free list more than once. "bad freelist" indicates the presence of bad blocks on the free list. A "bad" block is one that lies outside the bounds of the file system. A "dup" (duplicated) block is one associated with the free list and an i-node, or with more than one i-node. All the errors above *must* be corrected before the file system is mounted. "bad ifree" means allocated i-nodes are on the free i-node list; this is inconsequential.

NOTES
In earlier releases of COHERENT, **icheck** acted upon a default file system if none was specified.

NAME
if – conditional command execution

USAGE
if *sequence1*
then *sequence2*
[**elif** *sequence3*
then *sequence4*] ...
[**else** *sequence5*]
fi

DESCRIPTION
The shell **if** construct executes commands conditionally, depending on the exit status of the execution of other commands. First **if** executes the commands in *sequence1*. If the exit status is zero, it executes the commands in *sequence2* and terminates. Otherwise, it executes the optional *sequence3* if given, and executes *sequence4* if the exit status is zero; it executes additional **elif** clauses similarly. If the exit status of each tested command sequence is nonzero, it executes the optional **else** part *sequence5*. Because the shell recognizes a reserved word only as the unquoted first word of a command, each **then**, **elif**, **else** and **fi** must occur unquoted at the start of a line or preceded by ';'.

The shell executes **if** directly.

SEE ALSO
sh, **test**

NAME
join – join two databases

USAGE
join [**–a**[*n*]] [**–e** *string*] [**–j**[*n*] *keyf*] [**–o** *n.m* ...] [**–tc**] *file1 file2*

DESCRIPTION
join processes two text files *file1* and *file2* containing relational databases. If either file name is '–', the standard input is used for that file.

For the purposes of **join**, a database file contains a set of records, one per input line. Each record contains a number of *fields*. One field is differentiated as *key* field for each file. Each file must be sorted by key field, for example with **sort**.

By default, the key field is the first field in each record. The **–j** option changes the key field number to *keyf* for the desired file. In this and other options below, the optional file number *n* must be **1** to indicate *file1* or **2** to indicate *file2*. If no *n* is given, both *file1* and *file2* are assumed.

Normally fields are separated by any amount of white space (blanks or tabs). Leading blanks or tabs are not considered part of the fields. With the **–t** option, the separator character is *c*. With this option zero-length fields are possible; every occurrence of the separator ends the previous field and starts a new one.

Output consists only of records for which the key field occurs in both files. As a consequence of the sorted order of the input, the output is also sorted by the key field. Each output record has first the key field, then each field from the *file1* record but the key field, and then each field from the *file2* record but the key field. Fields are separated in the output with the specified field character, or with a space character if no **–t** option was given. Output records are always terminated with a newline. Under the **–e** option, *string* is printed for each empty field.

The **–a** option enables printing of records found in only file *n*. If *n* is missing, unpaired records are printed from both input files. To output only certain fields, the **–o** option precedes a list of desired

fields to print. Each element is of the form *n.m* where *n* is the file number and *m* is the field number.

For example,

```
join -t: -j1 3 -o 1.3 2.4 1.4 1.1 2.2 filea fileb
```

joins **filea** and **fileb** which have fields separated by the colon (':') character. The join field number is 3 for **filea** and 1 (by default) for **fileb**. The selected five fields are produced in the output.

SEE ALSO
awk, **comm**, **sort**, **uniq**

DIAGNOSTICS
join diagnostics are meant to be self-explanatory.

NAME
kermit – remote system communication and file transfer

USAGE
kermit c[bel *baud esc line* **]**
kermit r[bdfhilt *baud line* **]**
kermit s[abdfhilmtx *baud line* **]** *file* ...

DESCRIPTION
kermit allows the user to communicate with a remote computer system and to transfer files between the local and remote systems. **kermit** can transfer ASCII or binary files of any length in either direction. The two computers must be able to contact each other; for example, they might be connected by a serial line or by modems over a telephone line. The user must have login privileges on both systems and appropriate permissions in directories used for file transfer.

The **kermit** command line specifies a *mode*, followed without intervening spaces by optional *flags*, perhaps followed by additional arguments and *files*. The three possible *modes* are:

c Connect the two systems so they can communicate.
r Receive files from the other system.
s Send each *file* to the other system.

kermit normally uses a default communication line at a default baud rate; on most systems, the default line is **/dev/modem** and the default baudrate is 2400. It normally strips leading directory information from the pathname of each *file* it sends and converts the name to upper case; it converts the file name to lower case when receiving. The following *flags* modify its normal behavior.

a Specify complete pathnames for sending and receiving files; used only with s mode. The **a** flag requires filenames in pairs: first the file to be sent, then the receiving file. For example, the command

kermit sa /usr/joe/stuff.c /usr/tom/src/thing.c

sends the file **/usr/joe/stuff.c** but specifies its name as **/usr/tom/src/thing.c** for the receiving system. The target

directory must exist on the receiving system. The **a** flag implies the **f** and **x** flags described below.

b *baud* Set the baud rate to *baud*.

d Debug mode. Tells **kermit** to print messages (on the standard output, not the standard error) describing its actions.

e *esc* Change the escape character from the default '^' to the given *esc*; used only with **c** mode. The escape character marks commands to **kermit c** while it is running, as described below.

f Suppress filename case conversion.

h Host mode. Tells **kermit** to use the same line for file transfer and for communication; used with either **r** or **s** mode on the remote system. When used with the **h** flag, **kermit** resets the line modes properly when it completes a file transfer. If the **h** flag is not used, it will probably leave the remote system line in raw no-echo mode.

i Image mode. Tells **kermit** to send a full eight bit byte for each character; this is necessary for transferring binary (non-ASCII) files. If the **i** flag is used when sending, it should also be used on the receiving system.

l *line* Use *line* for the connection between the two systems. For example, the command

kermit clb /dev/tty50 1200

tells **kermit** to use line **tty50** at 1200 baud instead of the default line and baud rate.

m MacIntosh mode. Necessary when sending files to an Apple MacIntosh; used only with **s** mode.

t Tymnet mode. Allows Tymnet to keep up with file transmission.

x Allows the specification of a complete pathname for the receiving file; used only with **s** mode. For example, the command

kermit sx mydir/stuff

sends the file **mydir/stuff** to **mydir/stuff** on the receiving system. The target directory must exist on the receiving system and the user must have write permission in it.

kermit c recognizes two escape sequences. The default escape character '^' can be changed with the **e** flag, as noted above.

^c Exit from **kermit** and break the connection between the two systems.

^s Suspend **kermit** on the host system but do not hang up the line.

Unlike some file transfer protocols, **kermit** requires that the user invoke **kermit** on both the sending and receiving systems to transfer a file. As shown in the example below, the user normally uses **kermit c** to connect to the remote system, invokes **kermit** with the **h** flag in either send or receive mode on the remote system, types "^s" to suspend the local **kermit c**, and finally invokes **kermit** in receive or send mode on the local system.

The following example demonstrates the use of **kermit**. The example assumes the user is already logged in on the local system. The communication line is **/dev/al0** and runs at 300 baud. The user wants to transfer **locfile** to the remote system and **remfile** from the remote system. System names are in *italics* on the left, user input is in Roman, system responses are in **bold**, and remarks are in parentheses.

local	kermit clb /dev/al0 300	(connect to remote)
local	**kermit: connected...**	(type a carriage return)
remote	**Coherent login:**	(perform login procedure)
remote	kermit shi remfile	(send from remote)
remote	**)S~_@X#T**	(part of protocol, ignore)
remote	^s	(suspend local **kermit**)
local	**kermit: suspended.**	
local	kermit rilb /dev/al0 300	(receive on local)
local	**kermit: Receiving REMFILE as remfile**	
local	**kermit: done.**	
local	kermit clb /dev/al0 300	(connect again)

remote	kermit rhi	(receive on remote)
remote	^s	(suspend local **kermit**)
local	**kermit: suspended.**	
local	kermit silb /dev/al0 300 locfile	(send from local)
local	**kermit: Sending locfile as LOCFILE**	
local	**kermit: done.**	
local	kermit clb /dev/al0 300	(connect again)
remote	<Ctrl-D>	(log off the remote system)
remote	**Coherent login:**	
remote	^c	(disconnect local **kermit**)
local	**kermit: disconnected.**	

FILES
/dev/modem — default line

SEE ALSO
"Kermit: A File-Transfer Protocol for Universities," *BYTE*, June 1984 pp. 255 ff., July 1984 pp. 143 ff.

DIAGNOSTICS
kermit may print the following error messages:
Aborting with following error from remote host:
problem on receiving system.
Bad line speed: illegal baud rate.
Cannot create *name*: receiving system cannot create *name*.
Cannot open file *name*: sending system cannot open *name*.
Cannot open *line*: wrong *line* number.
No line specified for connection: *line* argument missing.
Receive failed: file not received.
Send failed: file not sent.
Speed setting not implemented: incorrect baud rate.
Yes, I'm still here...: connect command repeated.

NOTES
If you type **kermit c** and get the message "kermit connected" but the remote system does not respond, check the line connecting the two systems and the ability of the remote system to accept a login on the line.

Always remember to log off the remote system when done; **kermit** will not do it for you. If you do not log off, the next person connecting to the remote system over the same line will be logged in as you.

The file transfer protocol uses small (96 character) checksummed packets, with ACK/NAK responses from the receiving system. The timeout period is five seconds and **kermit** does ten retries before it abandons an attempted file transfer.

The **kermit** protocol was developed at the Columbia University Center for Computing Activities.

Tymnet is a trademark of Tymshare, Inc.

NAME
kill – kill a process

USAGE
kill [*–signal*] *pid* ...

DESCRIPTION
The COHERENT system identifies each active process by a unique process id, or *pid*. **kill** sends the specified *signal* to each process *pid* in the given list. The *signal* should be given by number, as defined in the file **/usr/include/signal.h** or **/usr/include/msig.h**. By default, *signal* is SIGTERM—software termination.

If the given *pid* is 0, **kill** signals each process started by the user from the same tty.

The shell **sh** prints the process id of a process if the command is detached. The **ps** command prints a list of all active processes, with process ids and command line arguments.

Only processes owned by the user can be killed; the superuser, however, can kill anything. A process cannot ignore or catch SIGKILL.

FILES
/usr/include/msig.h machine dependent signal numbers
/usr/include/signal.h machine invariant signal numbers

SEE ALSO
ps, sh
COHERENT System Manual: **getpid, init, kill, signal**

NAME
lc – categorize files in a directory

USAGE
lc[– **1abcdfp**] [*directory* ...]

DESCRIPTION
lc lists the names of the files under each argument *directory*, or the current directory if no *directory* is given. The files are categorized by filetypes (files, directories, and so on) and listed in columns within each category.

The following options modify the output.

– **1** List only one file name per line (do not columnize).

– **a** List all file names, including '.' and '..'.

– **b** List block special files only.

– **c** List character special files only.

– **d** List directories only.

– **f** List regular files only.

– **p** List pipe files only.

SEE ALSO
ls

NAME
ld – load/bind relocatable object files

USAGE
ld [*option* ...] *file* ...

DESCRIPTION
The result of a compilation or of an assembly is usually a *relocatable object file*. ld binds relocatable object files with libraries produced by ar to construct an executable file. ld is sometimes called a *linker*, a *link editor*, or a *loader*.

ld scans its arguments in order and interprets each *option* as described below. ld inspects a magic number at the front of each input *file* to determine if it is an object module or an archive (see ar.h and l.out.h in the *COHERENT System Manual*). ld rejects all other arguments with a diagnostic.

ld binds object files to the output file and adds their symbols to the symbol table. It scans libraries for modules satisfying undefined global references, and adds these (with their symbols) to the output file as necessary. Added library modules may themselves introduce further undefined globals; therefore ld scans the library repeatedly until it resolves no more undefined references. Thus the order of modules in a library has no significance (except to time required to load). However, the order of libraries in the argument list *is* significant. ranlib creates an index at the front of a library which can speed up the scanning considerably.

The available options are:

–d	Define common regions even if relocation information is retained. By default ld leaves common areas undefined if there are undefined symbols or if the –r option is specified.
–e *entry*	Specifies the *entry* point of the output module, either as a symbol or as an absolute octal address.
–i	Bind the output with separate instruction and data segments, each starting at location 0. By default all segments are concatenated.
–k[*system*]	Bind the output as a kernel process or loadable driver. The starting address depends on the target machine, and ld scans the *system* load file symbol table for currently undefined symbols. *system* defaults to /coherent.
–l*name*	An abbreviation for the library /lib/lib*name*.a or /usr/lib/lib*name*.a (if the first is not found).
–n	Bind the output with separate shared and private segments, and with each starting on a hardware segment boundary, so that several processes can use a single copy of the shared segment simultaneously. The –i option is implied by the –n option.
–o *file*	Write output to *file* (default: l.out).
–r	Retain relocation information in the output, and issue no diagnostic for undefined symbols. By default ld discards relocation information from the output if there are no undefined symbols.
–s	Strip the symbol table from the output. The same effect may be obtained by using strip. The –s and –r options are mutually exclusive.
–u *symbol*	Add the given *symbol* to the symbol table as a global reference, usually to force the loading of a particular library module.
–X	Discard local compiler-generated symbols (of the form 'L...').
–x	Discard all local symbols.

FILES

l.out	default output
/coherent	for –k option
/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries

SEE ALSO
ar, as, cc, ranlib, strip
COHERENT System Manual: ar.h, l.out.h

NOTES
By default, COHERENT allocates a fixed amount of stack to each process, usually 4K. This is in addition to the stack required for

the environment and arguments passed to the process and is enough for almost all processes; most processes use less than 1K of stack. Since stack space follows the BSSD segment in memory, adding space to BSSD increases the process stack size. For example, assembling the program

```
.bssd
.blkb 1024
```

and linking its object to a compiled program provides 1K of additional stack.

NAME

learn – computer-aided instruction

USAGE

learn [*–directory*] [*subject* [*lesson* [*speed*]]]

DESCRIPTION

learn is a computer-aided instruction program. Normally it is used to teach a student about various facilities available within the COHERENT system. The premise of **learn** is that hands-on experience with the commands and facilities of COHERENT will increase the student's comprehension of the system.

learn uses short lessons, or *scripts*, to teach the student about a subject. Lessons may include questions to the student, for which there may be right or wrong answers. Many scripts allow the student to interact with the system before formulating a response.

Typing the command **learn** causes the program to ask interactively which available *subject* is desired. The **learn** command line can also specify the desired *subject*. Subjects normally distributed with COHERENT include **files** (information about the file system) and **editor** (information about the editor **ed**).

A student who wishes to resume a prematurely terminated **learn** session can use the optional *lesson* argument. This argument should give the lesson name which **learn** typed out in the previous session.

The other available options are primarily for the **learn** script writer. The optional *speed* is a number between 0 and 10 which determines the path **learn** follows through its script. The *–directory* argument allows the script writer to create lessons in the specified *directory* rather than in the default directory **/usr/learn**.

The *learn User's Manual* describes how to use **learn** and how to write **learn** scripts in greater detail.

DIAGNOSTICS

Most diagnostics indicate errors in the script, and are intended for the script writer; these should never be seen by the naive student. However, many things can make a script fail unexpectedly, and this can lead to baffling situations. Diagnostics aimed at the student are more self-explanatory, and usually include an interactive request for remedial action.

FILES

/usr/learn	default learn directory
/usr/learn/play/u*	user's temporary directory
/usr/learn/lib/*/L*	lesson scripts
/usr/learn/log/*	normal logging information

SEE ALSO
help, man
learn User's Manual

NOTES
Because the scripts required for the **learn** command are quite large, they might not be included on COHERENT systems with insufficient disk space. As a result, the command might not work as expected on all systems.

NAME
lex – lexical analyzer generator

USAGE
lex [**–t**] [**–v**] [*file*]
cc lex.yy.c –ll

DESCRIPTION
Many programs process highly structured input according to given rules; compilers are a familiar example. Two of the most complicated parts of such programs are *lexical analysis* and *parsing* (also called *syntax analysis*). The COHERENT system includes two powerful tools called **lex** and **yacc** to assist the programmer with the construction of these parts of the program. **lex** converts a set of lexical rules to a lexical analyzer, and **yacc** converts a set of parsing rules to a parser. The output of **lex** may be used directly, or may be used by a parser generated by **yacc**.

lex reads a specification from the given *file* (or from the standard input if none), and generates a C function called **yylex()**. **lex** puts the generated function in the file **lex.yy.c**, or on standard output if the **–t** option is used. The **–v** option prints some statistics about the generated tables.

The *lex Lexical Generator Tutorial* describes **lex** in detail. In brief, the generated function **yylex()** matches portions of its input to one pattern (sometimes called a regular expression) from a set of rules, or *context*, and executes associated C commands. Unmatched portions of the input are copied to the output. **yylex()** returns end of file when input has been exhausted.

User replaceable routines are: **input()**, **output(*c*)** (macros, use **#undef**), **main()**, **error(...)** (args same as **printf**), and **yywrap()** (library). If an action is desired on end of file, such as arranging for more input, **yywrap()** should perform it, returning 0 to keep going.

A full **lex** specification has the following format:

Macro definitions, of the form:

```
name            pattern
```

Start condition declarations:

```
%S              NAME ...
```

Context declarations:

```
%C              NAME ...
```

Code to be included in the header section:

```
%{
anything
%}
<tab or space> anything
```

Rules section delimiter (must always be present):

```
%%
```

Code to appear at the start of yylex():

```
<tab or space> anything
```

Rules for initial context, in any of the forms:

```
rule            action;
rule            | (means use next action)
rule            {
<tab or space>action;
<tab or space>}
```

For each additional context:

```
%C              NAME
...rules for this context...
```

End of rules section delimiter:

```
%%
```

Code to be copied verbatim, such as user provided `input(), output(), yywrap(),` or other.

lex matches the longest string possible; if two rules match the same length string, the rule specified first takes precedence. lex puts the matched string, or *token*, in the char array **yytext[ ]**, and sets the variable **yyleng** to its length.

Actions may use the following:

ECHO	output the token
REJECT	perform action for lower precedence match
BEGIN *NAME*	set start condition to *NAME*
BEGIN 0	clear start condition
yyswitch(*NAME***)**	switch to context *NAME*, return current
yyswitch(0)	switch to initial context
yynext()	steal next character from input
yyback(*c***)**	put character *c* back into input

yyless(*n***)**	reduce token length to *n*, put rest back
yymore()	append next token to this one
yylook()	returns number of chars in input buffer

lex rules are contiguous strings of the form

[<*NAME*,...>] [^] *token* [*/lookahead*] [$]

where brackets [ ] indicate optional items.

<*NAME*,... >	match only under given start conditions
^	matches the beginning of a line
$	matches the end of a line
token	pattern for text of *token* to match
/lookahead	pattern for trailing text to match

Pattern elements:

a	the character **a**
\a	the character **a**, even if special
.	any character except newline
[abx–z]	any of **a**, **b**, or **x** through **z**
[^abx–z]	any except **a**, **b**, or **x** through **z**
abc	the string **abc**, even if any are special
{name}	the macro definition *name*
(exp)	the pattern *exp* (grouping operator)

Optional operators on elements:

e?	zero or one occurrence of *e*
*e**	zero or more consecutive *e*s
e+	one or more consecutive *e*s
e{*n*}	*n* (a decimal number) consecutive *e*s
e{*m*,*n*}	*m* through *n* consecutive *e*s

Patterns may be of the form:

e1e2	matches the sequence *e1 e2*
e1|*e2*	matches either *e1* or *e2*

lex recognizes the standard C escapes: \n, \t, \r, \b, \f, and *ooo* (octal representation). The special characters:

" \ ( ) < > { } % * + ? [–] ^ / $. |

must be prefixed with \ or enclosed in double quotes " " (excepting " and \) to be normal. Inside classes, only the characters . ^ – \ and] are special.

FILES
/usr/lib/libl.a

SEE ALSO
yacc
lex Lexical Generator Tutorial

NAME
ln – create a link to a file

USAGE
ln [**–f**] *oldfile* [*newfile*]
ln [**–f**] *oldfile ... directory*

DESCRIPTION
A *link* provides a mechanism for associating more than one file name with a single file. Changes to the file's attributes or data from any link will have the same effect; there is no distinction among links to the same file.

In its first form, **ln** makes a link from *oldfile* to *newfile*, provided *newfile* does not already exist. If *newfile* is omitted, a link is created in the current directory with the same file name as *oldfile*, with leading directory information removed.

In the second form, **ln** creates links in the specified *directory* with the same file names as each *oldfile*, with leading directory information removed.

If the new file name already exists, the **–f** option will force the command to work (by first unlinking the new file name).

Links to directories or across file systems are impossible.

SEE ALSO
cp, ls, mv, rm

Maintenance

NAME
load – load device driver

USAGE
/etc/load *file*

DESCRIPTION
load loads the binary device driver *file* into memory. The device files implicitly associated with the driver allow access to the desired hardware. By convention, device drivers are stored in the **/drv** directory.

load is restricted to the superuser.

The **uload** command unloads a device driver.

FILES
/drv/* driver files

SEE ALSO
uload
COHERENT System Manual: **sload**

NOTES
Because of hardware restrictions, the COHERENT system does not support loadable device drivers on systems based on the 8086 or 8088 processors (such as the IBM Personal Computer). The **/drv** directory and the **load** command do not exist on such systems.

NAME
login – log in or change user name

USAGE
login [*username*]

DESCRIPTION
The COHERENT system normally invokes **login** as part of the log in sequence on an unused terminal. The user may also invoke **login** directly from the shell **sh**, usually to change to a different user name. If *username* is not present, **login** prompts the user. If the account has a password, **login** asks for it.

If the user logs in successfully, **login** may print a message of the day. **login** notifies the user if mail is waiting to be read. **login** sets the working directory to the user's base directory and sets the user id and group id. Ownership of the tty is transferred to the user. The login accounting file is updated. Finally, if a program is specified in **/etc/passwd**, **login** reads **/etc/profile** for lines beginning "export" and inserts the remainder of the line into the environment; then **login** executes the specified program. If the program field is blank, **login** executes **sh**, which runs **$HOME/.profile** if present.

From the shell, a user may log in by typing **login** or by typing an end of file (normally <**ctrl-D**>) to terminate the previous shell.

FILES

/etc/passwd	user info
/etc/motd	message of the day
/etc/profile	system profile
/etc/utmp	current users on system
/usr/adm/wtmp	login accounting history
/usr/adm/failed	log of failed login attempts
$HOME/.profile	user profile

SEE ALSO
ac, sh, su
COHERENT System Manual: **getty, utmp.h**

NAME
look – find matching lines in a sorted file

USAGE
look [**–df**] *string* [*file*]

DESCRIPTION
look scans through the sorted *file* and prints each line which begins with an occurrence of the *string* specified.

Options may be used to specify the ordering relation used in the search:

–d Defines the ordering to be dictionary order; the only characters tested are alphanumerics and blanks.

–f Converts all alphabetic characters to upper case.

If no *file* is specified, **look** assumes **/usr/dict/words** with the **–df** option.

FILES
/usr/dict/words file of words (sorted with **sort –df**).

SEE ALSO
sort

NOTES
Because the word file **/usr/dict/words** is quite large, it might not be included on COHERENT systems with insufficient disk space. As a result, the command might not work as expected on all systems.

NAME
lpr – send to line printer spooler

USAGE
lpr [**–cmnr**] [**–b** *banner*] [*file* ...]

DESCRIPTION
lpr lets a user print each specified *file* on the line printer, without conflicting with printing by other users. If no *file* is specified, **lpr** prints the standard input on the line printer.

The following options exist:

b The next argument is the banner.

c Copy the files (allowing changes to be made before the printing completes).

m Send a message when the printing completes.

n Do not send a message (default).

r Remove the files when they have been spooled.

lpskip terminates or restarts the current listing.

FILES

/dev/lp	line printer
/usr/lib/lpd	line printer daemon
/usr/spool/lpd	spool directory
/usr/spool/lpd/dpid	daemon lockfile

SEE ALSO
lpskip, **pr**
COHERENT System Manual: **lpd**

NOTES
lpr writes a 80-column banner page before printing each *file*. On a single-user computer system, running a **pr** command in the background with its output redirected to the line printer device is usually preferable to using **lpr**.

NAME
lpskip – terminate/restart current line printer listing

USAGE
lpskip[–r]

DESCRIPTION
lpskip gives some control over printing with the line printer spooler. When invoked without the **–r** option, **lpskip** terminates the current listing with a message. When invoked with the **–r** option, **lpskip** restarts the current listing. This is useful when the paper runs out or jams.

lpr spools files to the line printer.

FILES

/usr/lib/lpd	line printer daemon
/usr/spool/lpd	spool directory
/usr/spool/lpd/dpid	daemon lockfile

SEE ALSO
lpr, pr
COHERENT System Manual: **lpd**

NAME
ls – list directories

USAGE
ls[–acdfgilrstu] [*file* **...]**

DESCRIPTION
ls prints a sorted list of information about each *file* specified. Normally, **ls** sorts by file name and prints only the name of each *file*. If a directory is specified, **ls** sorts and lists its contents (not including '.' and '..'). If no *file* is given, **ls** lists the contents of the current directory.

The following flags control how **ls** sorts and displays its output.

a Print all directory entries (including '.' and '..'). Normally, **ls** suppresses '.' and '..'.

c Use the *attribute change* time (ctime).

d Treat directories as if they were files.

f Force each argument to be treated as a directory. This disables the **–lrst** options and sorting and enables the **–a** option.

g Display group name rather than user name of owner; only applicable with **–l**.

i Print the i-number of each file.

l Print information in long format. Additional fields give mode bits, link count, owner, size in bytes and date. For special files, major and minor device numbers replace the size field.

r Reverse the sense of the sort.

s Print the size (in blocks) of each file.

t Sort by time, newest first.

u Use the *access* time (atime).

The date **ls** prints in the long list format and the **–t** sort is the *modification* time (mtime), unless the **–c** or **–u** option is used.

The mode field in the long list format consists of 10 characters. The first is one of:

- `-` regular file
- **b** block special file
- **c** character special file
- **d** directory
- **p** named pipe
- **x** bad entry (remove it immediately!)

The remaining nine characters are permission bits, in three sets of three characters each. The first set pertains to the owner of the file, the second to users from the owner's group, and the third to users from other groups. Each set may contain 3 characters from the following.

- **r** readable
- **w** writable
- **x** executable
- `-` no permission
- **s** set effective user id or group id on execution
- **t** shared text is sticky

FILES
/etc/passwd user names
/etc/group group names

SEE ALSO
lc
COHERENT System Manual: **chmod, stat**

NAME
m4 – macro processor

USAGE
m4 [*file* ...]

DESCRIPTION
m4 is a macro processor. It allows the user to specify search strings (macro names) and strings to replace them (definitions) with a great degree of generality. Macros may take arguments, written in a natural functional notation. **m4** includes powerful file manipulation, conditional decision making, substring selection and arithmetic capabilities. The *m4 User's Manual* describes **m4** in detail.

m4 reads each specified *file* in the given order, or the standard input if none, and writes to the standard output. The filename '–' indicates the standard input.

m4 copies input to output until it finds a potential *macroname*. A *macroname* is a string of alphanumerics (letters, digits, underscore) beginning with a non-digit and surrounded by non-alphanumerics. If **m4** does not recognize the *macroname*, it simply copies it to the output and continues processing. If **m4** recognizes the *macroname* and the next character is an open parenthesis '(', an *argument set* follows:

macroname(*arg1*,...,*argn*)

m4 collects the arguments by processing them in the same manner as other text (thus arguments may contain macro calls), and diverts the resulting output text into storage. **m4** stores up to nine arguments; any more will be processed but not saved. An argument set consists of strings of text separated by commas (commas inside quotes or parentheses do not terminate an argument), and must contain balanced unquoted parentheses. **m4** strips arguments of unquoted leading space (blanks, tabs, newlines).

m4 then removes the *macroname* and its optional argument set from the input stream, processes them, and replaces them in the input stream by the resulting value. The value becomes the next piece of text to be read.

Quotes, normally of the form ` ', inhibit *macroname* recognition. m4 strips off one level of quotes when it encounters them (quotes are nestable). Thus `macroname' does not get processed, but changed to *macroname* and passed on.

m4 determines the *value* of a user-defined macro by taking the text which constitutes the macro's *definition* and replacing any occurrence within that text of '$*n*' (where *n* is 0–9) with the text of the *n*th argument. Argument 0 is the *macroname* itself.

Predefined macros perform various functions.

changequote[([*openquote*],[*closequote*])]
Changes the quote characters. Missing arguments default to ` for open or ' for close. Quotes will not nest if they are defined to be the same character. Value is null.

decr[(*number*)]
Decrements given *number* (default: 0) by one and returns resulting value.

define(*macroname*,*definition*)
Defines or redefines any macro. If a predefined macro is redefined, its original definition is irrecoverably lost. Value is null.

divert[(*n*)]
Redirects output to output stream *n* (default: 0, the standard output). The standard output is 0, and 1–9 are maintained as temporary files. Any other *n* results in output being thrown away until the next **divert** macro. Value is null.

divnum Value is current output stream number.

dnl Delete to newline: removes all characters from the input stream up to and including the next newline. Value is null.

dumpdef[(*macronames*)]
Value is quoted definitions of all *macronames* specified, or names and definitions of all defined macros if no arguments.

errprint(*text*)
Prints *text* on standard error file. Value is null.

eval(*expression*)
Value is a number which is the value of evaluated *expression*. Recognizes, in order of decreasing precedence: parentheses, **, unary + −, * / %, binary + −, relations, and logicals. Arithmetic is performed in longs.

ifdef(*macroname*,*defvalue*,*undefvalue*)
Returns *defvalue* if *macroname* is defined and *undefvalue* if not.

ifelse(*arg1*,*arg2*,*arg3*...)
Compares *arg1* and *arg2*. If they are the same, returns *arg3*. If not, and *arg4* is the last argument, returns *arg4*. Otherwise, the process repeats, comparing *arg4* and *arg5*, and so on. Like other m4 macros, takes a maximum of nine arguments.

include(*file*)
Value is the entire contents of the *file* argument. If *file* is not accessible, a fatal error results.

incr[(*number*)]
Increments given *number* (default: 0) by one and returns resulting value.

index(*text*,*pattern*)
Value is a number corresponding to position of *pattern* in *text*. If *pattern* does not occur in *text*, value is −1.

len(*text*)
Value is a number corresponding to length of *text*.

maketemp(*text*)
Value is *text* with last six characters, usually **XXXXXX**, replaced with current process id and a single letter. Same as system call **mktemp**.

sinclude(*file*)
Value is the entire contents of the *file* argument. If the *file* is not accessible, returns null and processing continues.

substr(*text*[,*start*[,*count*]])
Value is a substring of *text*. The *start* position may be left-oriented (nonnegative) or right-oriented (negative). The *count* specifies how many characters to the right (if

positive) or to the left (if negative) to return. If absent, it is assumed to be large and of the same sign as *start*. If *start* is omitted, it is assumed to be 0 if *count* is positive or omitted, or −1 if *count* is negative.

sysemd(*command***)**
Passes the given *command* to the shell **sh** for execution. Value is null. Same as system call **system**.

translit(*text,characters*[,*replacements*])
Replaces given *characters* in *text* with the corresponding characters from *replacements*. If the *replacements* argument is absent or too short, null replaces selected *characters*. Value is *text* with specified replacements.

undefine(*macroname***)**
Removes macro definition. Value is null. If a predefined macro is redefined, its original definition is irrecoverably lost.

undivert[(*stream*[,...])]
Dumps each specified *stream* onto the current output stream. With no arguments, **undivert** dumps all output streams in numeric order. **m4** will not dump any output stream onto itself. At the end of processing, **m4** automatically dumps all diverted text to standard output in numeric order. Value is null.

SEE ALSO
COHERENT System Manual: **mktemp, system**
m4 User's Manual

NAME
mail − computer mail

USAGE
mail [**−mpqr**] [**−f** *file*] [*user* ...]

DESCRIPTION
mail provides a method of communication with other COHERENT system users. It is particularly useful for sending information to users not currently logged in. The **msg** and **write** commands are useful for communication with users who are logged in.

If any *user* is specified, **mail** reads a message from the standard input, prepends the date and the sender's name, and sends the result to each *user*. Either an end of file character (normally <ctrl-D>) or a line containing only the character '.' terminates the message. Each **user** will be greeted by the message "You have mail." when he or she next logs in. **mail** prefixes any line in the message that might be confused with the header by the character '>'.

If no *user* is given, **mail** reads the user's mail message by message. The following commands allow the user to save, delete, or send each message to another user interactively.

d	Delete the current message and print the next message.
m [*user* ...]	Mail the current message to each *user* given (default: yourself).
p	Print the current message again.
q	Quit, updating mailbox file to reflect changes.
s [*file* ...]	Save the current mail message with the usual header in each *file* (default: **mbox**).
t [*user* ...]	Send a message read from the standard input (terminated by end of file or by a line containing only '.') to each *user* (default: yourself).
w [*file* ...]	Write the current message without the usual header in each *file* (default: **mbox**).
x	Exit without updating the mailbox file.
newline	Print the next message.
−	Print the previous message.

EOF	Quit, updating mailbox; same as **q**.
?	Print a summary of available commands.
!*command*	Pass the given *command* to the shell for execution.

The following command line options control the sending and reading of mail.

−f *file*	Read mail from the given *file* instead of the default, which is **/usr/spool/mail/***user* for a given *user*.
−m	Send a message to the terminal of any *user* who is logged into the system when mail is sent. Normally, a user receives notification of incoming mail only when logging in.
−p	Print all mail without interaction.
−q	Quit without changing the mailbox if an interrupt character is typed. Normally, an interrupt character stops printing of the current message.
−r	Reverse the order of printing messages. Normally, **mail** prints messages in chronological order (oldest messages first), but under this option it prints messages in reverse chronological order.

FILES

dead.letter	message which **mail** could not send
mbox	default saved mail
/etc/passwd	user identities
/etc/utmp	logged in users
/tmp/mail*	temporary and lock files
/usr/spool/mail	mailbox directory, filed by user name

SEE ALSO

msg, write, xmail

NOTES

mail stores mail for a given *user* in file **/usr/spool/mail/***user*. *user* owns this file, and can therefore permit or deny access to the mail by other users.

Earlier releases of COHERENT included a different **mail** command.

NAME

make − program building discipline

USAGE

make [*option* ...] [*argument* ...] [*target* ...]

DESCRIPTION

make assists in building programs from more than one compilable module. Complex programs are often constructed of several *object modules*, which are the result of compiling *source programs*. Source programs may refer to **include** files, which are subject to independent change. Some programs may be generated from specifications, for example by **yacc** or other generators. Recompiling and relinking complicated programs correctly can be difficult and tedious.

make regenerates a program, based upon a specification of the structure of the program in **makefile** and the modification times of the files involved.

makefile has three types of lines: macro definitions, dependency definitions, and commands. Macro definitions contain the character =, dependency definitions have a target name at the beginning of a line followed by a colon, and command lines begin with a space or tab. Comments within lines begin with an unquoted **#** and end at the end of the line. Long non-comment lines may be broken with a quoted newline character. If no *target* is given on the command line, **make** assumes the target to be the first target in **makefile**.

Dependencies

makefile specifies which files depend upon other files and how to recreate the dependent files. Each *target* file is followed by a colon, followed by a space-separated list of files upon which it depends. The commands to recreate the dependent file are on the following lines, each beginning with a tab or space. If the target file **test.o** depends upon the source file **test.c**, the dependency is illustrated by

```
test.o: test.c
        cc -O test.c -o test.o
```

If **test.c** is modified or recreated, **make** will issue the **cc** command to regenerate the dependent file **test.o**.

make knows about common dependencies, such as that **.o** files depend upon **.c** files with the same basename. The target **.SUFFIXES** contains the suffixes **make** knows about. **make** also has a set of rules to regenerate dependent files. For example, for a source file with suffix **.c** and dependent file suffix **.o**, the target **.c.o** gives the regeneration rule:

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

Here **$<** stands for the name of the file causing the action. The default suffixes and rules can be changed.

Macros

To simplify the writing of complex dependencies, **make** provides a *macro* facility. To define a macro, write

NAME = *string*

The *string* is terminated by the end of the line, so it may contain blanks. To refer to the value of the macro, use a dollar sign '$' followed by the macro name enclosed in parentheses:

$(*NAME*)

If the macro name is one character, the parentheses are not necessary. **make** uses macros in the definition of default rules:

```
.c.o:
        $(CC) $(CFLAGS) -c $<
```

where the macros are defined as

```
CC=cc
CFLAGS=-O
```

Other built-in macros used in interpretation of rules are:

```
$*      target name less suffix
$@      target name
$<      list of referred files
$?      referred files newer than target
```


Each command line *argument* should be a macro definition of the form

```
"OBJECT=a.o b.o"
```

Arguments including spaces must be surrounded by quotes, since blanks are significant to the shell **sh**.

Options

−d (Debug) Give verbose printout of all decisions and information going into decisions.

−f *file* Specifies that *file* contains the **make** specification. If this option does not appear, **make** uses the file **makefile** or **Makefile** in the current directory. If *file* is '−', **make** uses the standard input.

−i Ignore error returns from commands and continue processing. Normally **make** exits if commands return error status.

−n Test only; suppresses actual execution of commands.

−p Print all macro definitions and target descriptions.

−q Return a zero exit status if the targets are up to date. No commands are executed.

−r Do not use built-in rules describing dependencies.

−s Do not print command lines when executing them. Commands preceded by '@' are not printed, except under the **−n** option.

−t (Touch) Force the dates of targets to be the current time, bypassing actual regeneration.

FILES

makefile	
Makefile	list of dependencies and commands
/usr/lib/makeactions	default actions
/usr/lib/makemacros	default macros

SEE ALSO

as, cc, ld, touch

DIAGNOSTICS
If the command **man** *title* fails, try **man −n** *title* instead.

NOTES
Because the data files required for the **man** command are quite large, they might not be included on COHERENT systems with insufficient disk space. As a result, the command might not work as expected on all systems.

Additional manual pages specific to certain versions of COHERENT are in different directories; the directory names are formed by prefixing a machine name to the usual directory name. For example, command manual pages for IBM PC COHERENT are in **/usr/man/ibmcmd** in raw form and in **/usr/man/ibmcmdman** in processed form.

NAME
mesg − permit/deny messages from other users

USAGE
mesg [y] [n]

DESCRIPTION
Normally a user can communicate with other currently logged in users with the **msg** and **write** commands. **mesg** lets the user allow or disallow messages from other users.

The argument **y** allows messages, while argument **n** disallows messages. With no argument, **mesg** prints the current state (as **yes** or **no**) without changing it.

FILES
/dev/*

SEE ALSO
msg, write

NOTES
The owner execute mode bit of the user's tty indicates whether messages are allowed.

NAME
mkdir – make a new directory

USAGE
mkdir *directory* ...

DESCRIPTION
mkdir creates each specified *directory*. Files or directories with the same name must not already exist. Each new **directory** will be empty except for the entries '.' (the directory's link to itself) and '..' (the link to its parent).

SEE ALSO
rm, rmdir

DIAGNOSTICS
mkdir returns 1 on error.

Maintenance

NAME
mount – mount file system

USAGE
/etc/mount [*special directory* [**–ru**]]

DESCRIPTION
mount mounts a file system from the block special file *special* onto *directory* in the system's directory hierarchy. This operation makes the root directory of the mounted file system accessible using the specified *directory* name.

If the **–r** option is specified, the file system is read only. This is useful for preventing inadvertant changes to precious file systems. The system will not update information such as access times if the **–r** option is used.

The **–u** option causes **mount** to write an entry into the mount table file **/etc/mtab** without actually performing the mount.

When invoked with no arguments, **mount** summarizes the mounted file systems and where they attach. The *root file system* is already mounted when COHERENT is booted and therefore does not appear in this list.

The **umount** command unmounts a previously mounted file system.

FILES
/etc/mtab mount table
/dev/*

SEE ALSO
check, mkfs, mknod, umount
COHERENT System Manual: **mount**

DIAGNOSTICS
Errors can occur if *special* or *directory* does not exist or if the user has no permissions on *special*.

The message

```
/etc/mtab older than /etc/boottime
```

indicates that **/etc/mtab** has probably been invalidated by rebooting the system.

Attempting to **mount** a block special file which does not contain a COHERENT file system may have disasterous consequences. **mkfs** must be used to create a file system on a blank disk or tape before it is mounted.

NAME
mkfs – make a new file system

USAGE
/etc/mkfs *filesystem proto*

DESCRIPTION
mkfs makes a new file system. The *filesystem* argument gives the name of the file (normally a block special file) where the new file system will reside. The contents of the newly created file system are described by the prototype file *proto*.

If *proto* is a number and not the name of a file, **mkfs** creates an empty file system (containing only a root directory) with size given by *proto*. The number of i-nodes is found as a percentage of this number.

The first line of the prototype file *proto* contains the name of a file containing the boot strap, which must fit into block 0 of the disk.

The second line contains two numbers. The first gives the size of the file system in blocks and the second gives the number of i-nodes in the file system. Since each file or directory requires one i-node, this number represents the limit on the number of files that may be created in the file system.

Next, lines beginning with '%b' describe bad blocks in the file system. There may be one or more block numbers on each line, separated by white space. The indicated blocks are allocated to the bad block file (i-node 1).

The remaining lines in the prototype file define the names, attributes, and contents of the files and directories in the file system. These lines are divided into fields separated by white space (blanks or tabs) as follows.

The first field gives the name of the file or directory to be created. This field is missing for the first line, which describes the root directory of the file system.

The mode of the file is described by the second field, which is 6 characters in length. The first character gives the file type, namely ordinary ('–'), directory ('d'), block special ('b'), or character special ('c') file. The second character is 'u' for set user id on execution and '–' otherwise. The third character is 'g' for set group id

on execution and '–' otherwise. The last three characters specify permissions in octal; for example, '644' specifies read and write permission for the owner, read permission for other users from the same group, and read permission for users from other groups.

If the above file type was a directory, subsequent files are recursively defined under that directory, until the current level of directory is terminated by a line containing a '$' character.

The next two fields specify the owner's numerical user id and group id.

The last field describes file contents. For a directory, it is not needed. For an ordinary file, it is the name of a COHERENT file which will be copied to the newly created file. For block or character special files there are two fields, specifying the major and minor device numbers.

The following example specifies a proto file for a (faulty) double density 8-inch floppy disk:

```
/usr/src/sys/pdp11/boot/ryboot
985 60
%b 55
%b 185 86
d--755 3 1
        coherent ---644 3 1 /coherent
        tmp     d--777 3 1
        $
        bin     d--755 3 1
                mail    -u-755 0 1 /bin/mail
        $
        dev     d--755 3 1
                tty30   c--644 0 1 3 0
                tty35   c--644 0 1 3 5
                mt0     b--600 0 1 12 0
        $
$
```


EXAMPLES
The command

```
/etc/mkfs /dev/fdd0 640
```

will create a file system on an 8 sector-per-track floppy disk in drive 0. If the floppy disk has 9 sectors-per-track, use the command

```
/etc/mkfs /dev/fd0 720
```

SEE ALSO
check, chmod, mount, restor

DIAGNOSTICS
Diagnostics occur for badly constructed *proto* files or for I/O errors on the file system.

NAME
mknod – make a special file or named pipe

USAGE
/etc/mknod *filename type major minor*
/etc/mknod *filename* **p**

DESCRIPTION
In the first form, **mknod** creates a *special file*, which provides access to a device by the *filename* specified. Special files are conventionally stored in the **/dev** directory.

The *type* can be one of **b** (for block special file) or **c** (for character special file). Block special files tend to be devices such as disks or magnetic tape, upon which COHERENT uses an elaborate buffering strategy. Character special files are unstructured (character at a time) devices such as terminals, line printers, or communications devices. Character special files may also be random access devices; this circumvents system buffering, allowing transfers of arbitrary size directly between the user and the hardware.

The *major* device number uniquely identifies a device driver to COHERENT. The *minor* device number is a parameter interpreted by the driver; it might specify the channel of a multiplexor or the unit number of a drive.

The caller must be the superuser.

In the second form, **mknod** creates a pipe with the given *filename*. Named pipes can be used for communication between processes.

SEE ALSO
mount
COHERENT System Manual: **mknod**

FILES
/dev/*

NAME
mount – mount file system

USAGE
/etc/mount [*special directory* [**–ru**]]

DESCRIPTION
mount mounts a file system from the block special file *special* onto *directory* in the system's directory hierarchy. This operation makes the root directory of the mounted file system accessible using the specified *directory* name.

If the **–r** option is specified, the file system is read only. This is useful for preventing inadvertant changes to precious file systems. The system will not update information such as access times if the **–r** option is used.

The **–u** option causes **mount** to write an entry into the mount table file **/etc/mtab** without actually performing the mount.

When invoked with no arguments, **mount** summarizes the mounted file systems and where they attach. The *root file system* is already mounted when COHERENT is booted and therefore does not appear in this list.

The **umount** command unmounts a previously mounted file system.

FILES
/etc/mtab mount table
/dev/*

SEE ALSO
check, mkfs, mknod, umount
COHERENT System Manual: **mount**

DIAGNOSTICS
Errors can occur if *special* or *directory* does not exist or if the user has no permissions on *special*.

The message

```
/etc/mtab older than /etc/boottime
```

indicates that **/etc/mtab** has probably been invalidated by rebooting the system.

Maintenance

Attempting to **mount** a block special file which does not contain a COHERENT file system may have disasterous consequences. **mkfs** must be used to create a file system on a blank disk or tape before it is mounted.

NAME
msg – send a brief message to other users

USAGE
msg *user*
msg *user ... message*

DESCRIPTION
msg sends a short message to other COHERENT users currently logged in. Spooling daemons also use **msg** to notify users of completion of spooled requests.

In the first form above, **msg** reads one line from the standard input and sends it to the specified *user*. In the second form, **msg** reads the *message* argument and sends it to the list of *users*. If *message* is '–', the message is one line read from the standard input.

The **mesg** command can prevent other users from sending messages to the user. Normally, other users are permitted to send messages.

FILES
/etc/utmp information on logged in users
/dev/tty*

SEE ALSO
mail, mesg, who, write

NAME
mv – move/rename files and directories

USAGE
mv [**–f**] *oldfile newfile*
mv [**–f**] *file ... directory*

DESCRIPTION
mv renames (moves) files. In the first form above, **mv** renames *oldfile* to *newfile*. If *newfile* previously existed, **mv** deletes its former contents; if not, **mv** creates it. If *newfile* is a directory, **mv** places *oldfile* under that directory.

In the second form, **mv** renames each *file* so that it resides under *directory*.

If a file with the new name exists but is unwritable, **mv** will not delete it unless the **–f** option is specified.

mv will not copy directories between devices and will not remove directories which occupy the destination of the command.

If both the source and destination names are linked to the same file, no move is performed.

Normally, **mv** creates a link to the old file with the new name and then removes the old name. If it cannot create the link (for example, because the new file is on a different file system than the old), **mv** performs a copy and then removes the old file.

SEE ALSO
cp, ln

NOTES
mv tests the validity of directory moves by means of search permission. The superuser always has search permission and thus can use **mv** incorrectly.

Maintenance

NAME
ncheck – print file names corresponding to i-numbers

USAGE
ncheck [**–i** *number ...*] [**–as**] *filesystem ...*

DESCRIPTION
An *i-number* is a number which identifies an i-node. **ncheck** generates a list of file names by i-number for each *filesystem* specified, which should be the name of a device special file containing a proper COHERENT file system. Using the raw device generally decreases the time **ncheck** takes.

The output is in the unsorted traversal order of the file system hierarchy. **ncheck** distinguishes directories from files by suffixing '/.' to directory names.

Under the **–i** option, **ncheck** prints the file name corresponding to each i-number *number ...* in the given list. Under the **–a** option, **ncheck** prints the names '.' and '..' which it normally suppresses. Under the **–s** option, **ncheck** only prints the names of special files and set user ID mode files; this option allows the system administrator to quickly ascertain whether these files represent possible security breaches.

SEE ALSO
check, dcheck, icheck, quot

DIAGNOSTICS
ncheck appends '??' to the generated file name if it cannot find the proper parent structure while retrieving the file name information. It represents any loops detected in a file name by the characters '...'. Extremely addled file systems may draw other reasonably self-explanatory diagnostics.

NAME
newgrp – change to a new group

USAGE
newgrp *group*

DESCRIPTION
newgrp changes the user's group identification to the specifed *group*, if access is permitted. The file **/etc/group** determines group access. Group access may be unrestricted, or open to all users with specific exceptions, or restricted to certain users via a password.

The shell **sh** executes **newgrp** directly.

FILES
/etc/group

SEE ALSO
sh
COHERENT System Manual: **group**

DIAGNOSTICS
If **newgrp** succeeds, no diagnostic is printed.

NOTES
Interruption of **newgrp** can result in the user being logged out.

Maintenance

NAME
newusr – add new user to COHERENT system

USAGE
/etc/newusr *user* ...

DESCRIPTION
newusr adds each given *user* to the user base of the COHERENT system. It adds an entry for each *user* to the password file **/etc/passwd** and creates a home directory **/usr/***user* for each.

Only the superuser can add new users to the system with **newusr**.

FILES
/etc/passwd
/usr/*user*

SEE ALSO
passwd
COHERENT System Manual: **passwd**

DIAGNOSTICS
newusr complains if an entry for *user* already exists in the password file.

NOTES
newusr is implemented as a shell command file, so it can be changed easily to specify a different home directory pathname or a different group id.

NAME
nm – print name list

USAGE
nm [–adgnopru] [*file* **...]**

DESCRIPTION
nm prints the name list (symbol table) of each object *file* in its argument list. If no *file* is specified, **nm** prints the name list of file **l.out**.

If an argument *file* is an archive, **nm** prints a name list for each of the archive's members.

The symbols are normally sorted alphabetically. Each symbol is preceded by its value (spaces if the symbol is undefined) and its type, designated as follows:

SI	shared instructions,
PI	private instructions,
BI	instruction space BSS,
SD	shared data,
PD	private data,
BD	data space BSS,
D	debug table,
A	absolute, and
C	common block.

The type indicator is in lower case if the symbol is local and in upper case if the symbol is global.

Options are:

–a Print all symbols. Normally **nm** prints names in a C style format and ignores any symbol with a name inaccessible from C language programs.

–d Print only symbol definitions.

–g Print only global symbols.

–n Sort numerically rather than alphabetically.

–o Suppress titles. Prepend the file or archive member name to each output line.

–p Print symbols in symbol table order.

–r Sort in reverse order.

–u Print only undefined symbols.

SEE ALSO
ar
COHERENT System Manual: **ar.h, l.out.h**

NAME
nroff – text processor

USAGE
nroff [*option* ...] [*file* ...]

DESCRIPTION
nroff processes each given *file*, or the standard input if none is specified, and prints the formatted result on the standard output. The input must contain formatting instructions as well as the text to be processed.

The *nroff Text Processor Tutorial* provides a detailed introduction to **nroff** facilities and a summary of commands. In brief, basic commands provide for such things as setting line length, page length and page offset, generating vertical and horizontal motions, indentation, filling and adjusting output lines, centering, and so on. The great flexibility of **nroff** lies in its acceptance of user-defined macros to control almost all higher-level formatting. For example, the formation of paragraphs, header and footer areas, and footnotes must all be implemented by the user via macros.

Options may be listed in any order. The options are:

–d Debugging mode.

–i Read from the standard input after reading the given *files*.

–m*name*
Include the macro file **/usr/lib/tmac.***name* in the input stream.

–n*N* Number the first page of output *N*.

–r*aN* Set number register *a* to the value *N*.

–x Do not eject to the bottom of the last page when text ends. This is often useful if the output device is a CRT.

FILES
/tmp/rof* temporary files
/usr/lib/tmac.* standard macro packages

SEE ALSO
col, deroff
COHERENT System Manual: **man, ms**
nroff Text Processor Tutorial

NOTES
The **–ms** macros are sufficient for most ordinary manuscript processing. The uninitiated should become familiar with **nroff** through this macro package rather than trying to deal at once with the basic program. The **–man** macros format manual information for this manual, the *COHERENT System Manual*, and the **man** command.

NAME
od – file dump

USAGE
od [**–bcdox**] [*file*] [[**+**] *offset*[**.**][**b**]]

DESCRIPTION
od (for octal dump) prints the specified *file* as a sequence of bytes or machine words. If no *file* is specified, the '+' must be present and **od** dumps the standard input.

The output format is selectable:

b bytes printed in the default base,
c bytes printed as ASCII characters,
d words printed in decimal,
o words printed in octal, and
x words printed in hexadecimal.

The default base is octal on the PDP-11 and hexadecimal on the i8086 and Z-8001.

Dumping can start at a given *offset* into the file. The specified *offset* is octal unless the '.' suffix is present to signify decimal. The *offset* is in bytes unless the **b** suffix is present to signify 512-byte blocks.

SEE ALSO
db, scat, conv
COHERENT System Manual: **ascii**

NOTES
Clumsy.

NAME
passwd – set/change login password

USAGE
passwd [*user*]

DESCRIPTION
passwd sets or changes the password for the specified *user*. If *user* is not specified, **passwd** changes the password of the caller.

passwd requests that the old password (if any) be typed, to ensure the caller is the real McCoy. Next it requests a new password, and then requests it again in case of typing errors. **passwd** will ask for a longer password if the one given is too short or not unusual enough.

passwd uses the DES encryption routine **crypt** to perform encryption.

FILES
/etc/passwd encrypted passwords

SEE ALSO
login
COHERENT System Manual: **crypt**

NAME
prof – print execution profile

USAGE
prof [–abcs] [*progfile* [*monfile*]]

DESCRIPTION
prof interprets the profile file produced by an execution of a C program and reports the execution frequencies of each routine. It also reports the percentage of execution time spent in each routine.

prof normally reports times and frequencies spent for regions of programs between externally defined names. The *progfile* is the executable program; if omitted, **l.out** is assumed. The *monfile* is the monitor file produced during execution of the program; if omitted, **mon.out** is assumed.

To produce the **mon.out** file, a program must be compiled with the **–p** option to **cc**. To profile all modules, each module must be compiled with this option.

The following options are available.

–a Profile all symbols rather than just externals.

–b Print all bin information.

–c Print all call information.

–s Report stack usage high water mark.

FILES
l.out program file (with name list intact)
mon.out raw execution profile

SEE ALSO
cc, ld, nm
COHERENT System Manual: **profil**

NAME
ps – print process status

USAGE
ps [–afglmnrtwx] [–c *sys*] [–k *mem*]

DESCRIPTION
ps prints information about a process or processes. It prints the information in fields, followed by the command name and arguments. The fields include:

TTY The controlling terminal of the command, printed in short form. "44:" means **/dev/tty44** and "??:" means there is no controlling terminal.

PID Process id; necessary to know when the process is to be killed.

GROUP PID of the group leader of the process; the shell started up when the user logs in.

PPID PID of the parent of the process; very often a shell.

UID User id or name of the owner.

K Size of the process in kilobytes.

F Process flag bits:
- **01** means in core,
- **02** means locked in core,
- **04** means process is not executing,
- **10** means tracing is enabled.

S State of the process:
- **R** means ready to run (waiting for CPU time),
- **S** means stopped for other reasons (I/O completion, pause, etc.),
- **T** means being traced by another process,
- **W** means waiting for an existent child,
- **Z** means zombie (dead, but parent not waiting).

EVENT The condition which the process is anticipating; not applicable if the process is ready to run.

CVAL SVAL IVAL RVAL
Scheduling information; bigger is better.

UTIME Time consumed while running in the program (in seconds).

STIME Time consumed while running in the system (in seconds).

Normally, **ps** displays the TTY and PID fields of each active process started on the caller's terminal, as well as the command name and arguments. The following flags can alter this behavior.

a Display information about processes started from all terminals.

c The next argument *sys* gives the system executable image (default: **/coherent**). The namelist is searched for table addresses.

f Blank fields have '–' place-holders. This enables field-oriented commands like **sort** and **awk** to process the output.

g Print the group leader field GROUP if the **l** option is given.

l Long format. In addition to the TTY and PID fields, prints the PPID, UID, K, F, S and EVENT fields.

k The next argument *mem* is the memory file (default: **/dev/mem**).

m Print the scheduling fields CVAL, SVAL, IVAL and RVAL.

n Suppress the header line.

r Print the real size of the process, which includes the user and auxiliary segments assigned to the process. Since the user segment (usually 1 kilobyte) is shared by all processes owned by the user, this may give a misleading total size for all the user's processes.

t Print elapsed CPU time fields UTIME and STIME.

w Wide format output; print 132 columns instead of 80.

x Display processes which do not have a controlling terminal (e.g. the swapper).

FILES

/coherent default system file
/dev/mem default memory file
/dev/tty* list of terminal names

SEE ALSO

kill, size, wait
COHERENT System Manual: **mem**

NOTES

Each process can modify or destroy its command name and arguments. The state of the system changes even as **ps** runs.

NAME
pwd – print current working directory

USAGE
pwd

DESCRIPTION
pwd prints the name of the current working directory for the current shell process.

SEE ALSO
cd, sh

NAME
quot – summarize file system usage

USAGE
quot [–c] [–f] [–n] [–t] ***filesystem*** **...**

DESCRIPTION
quot produces several different summaries about the ownership of files for each ***filesystem*** argument given. When no options are specified, **quot** produces a two column listing giving the amount of space used by each user, sorted in decreasing order of filespace use; the first column gives the number of blocks used and the second gives the user name. Space used is always given in blocks.

Options are available to modify the normal output or specify a completely different action.

–c Gives a three-column breakdown of files by size. The first column contains all file sizes, in increasing order. The second column gives the number of files of the size indicated in the first. The third gives a cumulative sum of the sizes of all files less than or equal to the current size.

–f Adds an initial column containing the number of files to the front of the normal output.

–n Takes as input a list of i-numbers and file names, one per line and sorted in ascending order by i-number; ignores all lines not in this form. The output is in two columns, the first giving the owner and the second containing the file name for each entry in the input. This conforms to usage with the following pipeline:

```
ncheck filesystem | sort +0n | quot -n filesystem
```

–t Adds a line at the end containing totals to the normal output.

quot runs much faster with a raw device for ***filesystem***.

FILES
/etc/passwd

SEE ALSO
ac, du, ncheck, sa, sort

NOTES
Sparse files are recorded as if they had all of their blocks allocated.

In earlier releases of COHERENT, **quot** acted upon a default file system if none was specified.

NAME
ranlib – create index for library

USAGE
ranlib *library* ...

DESCRIPTION
ranlib makes each given *library* created by **ar** faster for **ld** to search. It inserts at the front a sorted index of all global symbols defined in object modules in the library, together with the offset of the module in the library. If the index already exists **ranlib** updates it.

FILES
__.SYMDEF index module

SEE ALSO
ar, ld
COHERENT System Manual: **ar.h**

DIAGNOSTICS
ranlib issues appropriate messages for I/O errors or bad format files. It does not rewrite a library until the last possible moment, so the library is usually unchanged in case of error. **ranlib** processes each library independently. The exit status is the number of libraries in which errors were encountered.

NOTES
This function should be subsumed by **ar**.

NAME
read – assign values to shell variables

USAGE
read *name* ...

DESCRIPTION
read reads a line from the standard input. It assigns each token of the input to the corresponding shell variable *name*. If the input contains fewer tokens than the number of names specified, **read** assigns the null string to each extra variable. If the input contains more tokens than the number of names specified, **read** assigns the last *name* in the list the remainder of the input.

The shell **sh** executes **read** directly.

SEE ALSO
sh

DIAGNOSTICS
read returns an exit status of 0 unless it encounters end of file or is interrupted while reading the standard input, in which case it returns 1.

NAME
readonly – mark shell variables as read only

USAGE
readonly [*name* ...]

DESCRIPTION
readonly marks each given shell variable *name* as a read only variable. The shell **sh** will not allow subsequent assignments to the variable.

With no arguments, **readonly** prints the name and the value of each shell variable currently marked as read only.

The shell executes **readonly** directly.

SEE ALSO
sh

NAME

scat – segmented concatenation

USAGE

scat [[*option* ...] [*file* ...]] ...

DESCRIPTION

scat prints each *file* on the standard output, a page at a time if the output is a tty. The standard input is read if no *file* is given. The text is processed to allow convenient viewing from a CRT; options described below select the nature of the processing. Option arguments begin with '–' and may be interspersed with file names.

scat scans two argument lists. The first is a string called SCAT in the environment. It should consist of arguments separated by white space (space, tab and newline characters), with no quoting or shell metacharacters. This string is a useful place to set terminal-dependent parameters (such as page width and length) and to place invoke lists (see below). The second argument list is supplied on the command line.

The options are:

–b*n* Begin output at input line *n*.

–c Represent all control characters unambiguously. Print control characters in the range 0–037 as a char in the range 0100–0137 prefixed by '^', so SOH appears as "^A". Print DEL as "^?". Prefix mark-parity characters (0200–0377) with '~', so mark-parity 'A' and SOH appear as "~A" and "~^A", respectively. Prefix the characters '^', '~' and '\' with '\'. This option overrides –t.

–cs Like –c, but map space ' ' to underscore '_', and prefix underscore '_' with '\'.

–ct Like –c, but map tabs to spaces, not "^I".

–i*n* Shift the display window right *n* columns into the text field. This is useful for viewing long lines.

–l*n* Set the display window length to *n* lines. The default is 24 normally, 34 for the Tek 4012.

–n Number input lines; wrapped lines are not numbered.

–r Remote; the output is not paged.

–s Squash empty lines.

–S*n* Seek *n* bytes into input before processing.

–t Truncate long lines. Normally scat wraps each long line, with the interrupted portion delimited by a '\'.

–w*n* Set the display window width to *n* columns. The default is 80 normally, 72 for the Tek 4012.

–x Expand tabs.

–.*suffix*
Invoke options by file name suffix. If a file name ends with .*suffix* then scat scans the argument sublist starting immediately after the invoke flag. New options will apply to the invoking file only. A sublist is terminated by the end of the argument list, by a file name, by the "– –" flag, or by another "–." (invokes do not nest).

– – Terminate a sublist (see previous option).

Numbers may begin with 0 to indicate octal, and may end in b or k to be scaled by 512 or 1024, respectively.

If the output is being paged, scat waits for a user response, which may be one of the following:

newline	Display next page.
/	Display next half-page.
space	Display next line.
f	Print current file name and line number.
n	scat next file.
q	Quit.

The following example shows how to use the environment argument list, invokes and sublists:

```
SCAT="-124 -.c -n -.s -b5"
export SCAT
scat *.c *.s
```

After processing the SCAT argument list, scat processes the command line argument list "*.c *.s" with the page length at 24 lines.

If a file is a C source ("*.c") the invoke option in the SCAT argument list numbers the output lines. If a file is an assembly source ("*.s") **scat** skips the first 4 lines.

SEE ALSO
cat, pr

NOTES
256 is a limit in several places.

NAME
sed – stream editor

USAGE
sed [**–n**] [**–e** *commands*] [**–f** *script*] ... *file* ...

DESCRIPTION
sed is a non-interactive text editor. It reads input from each *file* given, or from the standard input if none. It edits the input according to commands given in the *commands* argument and the *script* files. **sed** writes edited text to the standard output.

sed is similar to the interactive editor **ed**, but its operation is fundamentally different. **sed** normally edits one line at a time, so it may be used to edit very large files. After it constructs a list of commands from its *commands* and *script* arguments, **sed** reads its input one line at a time into a *work area*. Then **sed** executes each command which applies to the line, as explained below. Finally it copies the work area to the standard output (unless the **–n** option is specified), erases the work area, and proceeds with the next input line.

Line Identifiers
sed identifies input lines by integer line numbers, beginning with 1 for the first line of the first *file* and continuing through each successive *file*. Several special forms identify lines. These forms are:

n	A decimal number *n* addresses the *n*th line of the input.
.	A period '.' addresses the current input line.
$	A dollar sign '$' addresses the last line of input.
/pattern/	A *pattern* enclosed in slashes addresses the next input line containing an occurrence of the *pattern*. Patterns, also called regular expressions, are described in detail below.

Commands
Each **sed** command must be on a separate line. Most commands may be optionally preceded by a line identifier (abbreviated as [*n*] in the command summary below) or by two line identifiers separated by a comma (abbreviated as [*n*[,*m*]]). If no line identifier precedes a command, **sed** applies the command to every input line. If one line identifier precedes a command, **sed** applies

the command to each input line selected by the identifier. If two line identifiers precede a command, sed begins to apply the command when an input line is selected by the first, and continues applying it through an input line selected by the second.

The following commands are available.

[*n*]= Output the current input line number.

[*n*[,*m*]]!*command*
Apply *command* to each line *not* identified by [*n*[,*m*]].

[*n*[,*m*]]{*command*...}
Execute each enclosed *command* on the given lines.

:*label* Define *label* for use in a branch or test command.

[*n*]a\ Append new text after given line. New text is terminated by any line not ending in '\'.

b[*label*] Branch to *label*, which must be defined in a ':' command. If *label* is omitted, branch to end of command script.

[*n*[,*m*]]c\
Change specified lines to new text and proceed with next input line. New text is terminated by any line not ending in '\'.

[*n*[,*m*]]d Delete specified lines and proceed with next input line.

[*n*[,*m*]]D Delete first line in work area and proceed with next input line.

[*n*[,*m*]]g Copy secondary work area to work area, destroying previous contents.

[*n*[,*m*]]G Append secondary work area to work area.

[*n*[,*m*]]h Copy work area to secondary work area, destroying previous contents.

[*n*[,*m*]]H Append work area to secondary work area.

[*n*]i\ Insert new text before given line. New text is terminated by any line not ending in '\'.

[*n*[,*m*]]l Print selected lines, interpreting non-graphic characters.

[*n*[,*m*]]n Print the work area and replace it with the next input line.

[*n*[,*m*]]N Append next input line preceded by a newline to work area.

[*n*[,*m*]]p Print work area.

[*n*[,*m*]]P Print first line of work area.

[*n*]q Quit without reading any more input.

[*n*]r *file* Copy *file* to output.

[*n*[,*m*]]s[*k*]/*pattern1*/*pattern2*/[g][p][w *file*]
Search for *pattern1* and substitute *pattern2* for *k*th occurrence (default: first). If optional g is given, substitute all occurrences. If optional p is given, print the resulting line. If optional w is given, append the resulting line to *file*. Patterns are described in detail below.

[*n*[,*m*]]t[*label*]
Test if substitutions have been made. If so, branch to *label*, which must be defined in a ':' command. If *label* is omitted, branch to end of command script.

[*n*[,*m*]]w *file*
Append lines to *file*.

[*n*[,*m*]]x Exchange the work area and the secondary work area.

[*n*[,*m*]]y/*chars*/*replacements*/
Translate characters in *chars* to the corresponding characters in *replacements*.

Patterns

Substitution commands and search specifications may include *patterns*, also called *regular expressions*. Pattern specifications are identical to those of ed, except that the special characters '\n' match a newline character in the input.

A non-special character in a pattern matches itself. Special characters include the following.

^ Matches beginning of line, unless it appears immediately after '[' (see below).

$ Matches end of line.

\n Matches the newline character.

. Matches any character except newline.

***** Matches zero or more repetitions of preceding character.

[*chars***]** Matches any one of the enclosed *chars*. Ranges of letters or digits may be indicated using '–'.

[^*chars***]** Matches any character *except* one of the enclosed *chars*. Ranges of letters or digits may be indicated using '–'.

*****c Disregard special meaning of character *c*.

\(*pattern***\)**
Delimit substring *pattern*; for use with *****d described below.

In addition, the replacement part *pattern2* of the substitute command may also use the following:

& Insert characters matched by *pattern1*.

*****d Insert substring delimited by *d*th occurrence of delimiters '\(' and '\)', where *d* is a digit.

Options
The following options are available.

–e Next argument gives commands.

–f Next argument gives file name of command script.

–n Output lines only when explicit **p** or **P** commands are given.

SEE ALSO
ed

NAME
set – set shell option flags and positional parameters

USAGE
set [–ceiknstuvx [*name* **...]]**

DESCRIPTION
set changes the options of the current shell **sh** and optionally sets the values of positional parameters. The shell variable '$–' contains the currently set shell flags. If the optional *name* list is given, **set** assigns the positional parameters **$1, $2...** to the given shell variables.

The options are:

–c *string*
Read shell commands from *string*.

–e Exit on any error (command not found or command returning nonzero status) if the shell is not interactive.

–i The shell is interactive, even if the terminal is not attached to it; print prompt strings. For a shell reading a script, ignore signals **SIGTERM** and **SIGINT**.

–k Place all keyword arguments into the environment. Normally, the shell places only assignments to variables preceding the command into the environment.

–n Read commands but do not execute them.

–s Read commands from the standard input and write shell output to the standard error.

–t Read and execute one command rather than the entire file.

–u If the actual value of a shell variable is blank, report an error rather than substituting the null string.

–v Print each line as it is read.

–x Print each command and its arguments as it is executed.

– Cancel the **–x** and **–v** options.

The shell executes **set** directly.

SEE ALSO
sh

NAME

sh – command language interpreter

USAGE

sh [–ceiknstuvx] *token* ...

DESCRIPTION

sh, called the *shell*, is the default COHERENT command language interpreter. Other command languages can be provided on a per-user basis. The *sh Command Language Tutorial* describes the shell in detail.

The shell reads commands from the terminal or from a file and interprets them. A command may be either a program or a text file containing other commands. Shell constructs provide control flow logic. Commands can contain *patterns*, which the shell expands into file names. The shell can redirect input and output.

Commands

A command consists of a command name and optional command arguments, called *tokens*. A token is a string of graphic characters separated by spaces or tabs. Normally, the first token in a command is the command name.

Commands are combined with the pipe operator '|' to form *pipelines*. In the pipeline

```
a | b
```

the shell connects the standard output of **a** to the standard input of **b**. The shell runs each command in the pipeline as a separate process, and waits for the last command to finish before continuing.

Commands and pipelines can be joined into a *sequence* with the tokens ';', '&', '&&', and '||', in addition to newlines. The shell executes commands or pipelines separated by newlines or by ';' sequentially. For example,

```
a | b ; c | d
```

first executes the pipeline **a | b** and then executes the pipeline **c | d**. The shell executes any command followed by '&' asynchronously as a background (or detached) process and prints its process id. The shell executes the command following the token '&&' only if the preceding command returns a zero exit status, signifying success.

Similarly, it executes the command following '||' only if the preceding command returns a nonzero exit status, signifying failure. Newline, ';' and '&' bind less tightly than '&&' and '||'; the shell parses command lines from left to right if separators bind equally.

I/O Redirection

The *standard input*, *standard output* and *standard error* streams are normally connected to the terminal. A pipeline attaches the output of one command to the input of another command. Additionally, the operators '>', '>>', '<' and '<<' redirect the standard output and the standard input.

Output redirection sends standard output to *file* rather than to the terminal:

```
>file
```

creates *file* if it does not exist, and destroys its previous contents if it does exist. The operator

```
>>file
```

appends standard output to an existing *file*, or creates *file* if it does not exist.

In input redirection,

```
<file
```

accepts standard input from *file* rather than from the terminal. The input redirection operator

```
<<token
```

accepts standard input from the shell input until the next line containing only *token* in the shell input. The shell input between *token*s is called a *here document*. The shell will perform parameter substitution on the here document unless the leading *token* is quoted; parameter substitution and quoting are described below.

The standard input and output may also be redirected to duplicate other file descriptors. The operator '<&*n*' duplicates the standard input from file descriptor *n*, and '>&*n*' duplicates the standard output. The operators '<&−' and '>&−' close the standard input and output.

Other descriptors may be redirected by preceding the '<' or '>' with the digit of the descriptor to be redirected. For example, 2>&1 redirects file descriptor 2 (the standard error) to file descriptor 1 (the standard output). The system call **dup** (see the *COHERENT System Manual*) performs file descriptor duplication.

Each command executed as a foreground process inherits the file descriptors and signal traps (described below) of the invoking shell, modified by any specified redirection. Background processes take input from the null device **/dev/null** (unless redirected) and ignore interrupt and quit signals.

File Name Patterns

The shell interprets an input *token* containing any of the special characters '?', '*', or '[' as a file name *pattern*. The question mark '?' matches any single character except newline. The asterisk '*' matches a string of non-newline characters of any length (including zero). Square brackets '[ ]' enclose alternatives to match a single character; as in **ed**, ranges of characters can be separated by '−'. The slash '/' and leading period '.' must be matched explicitly in a pattern. The shell generates an alphabetized list of file names matching the pattern to replace the *token*. It passes the *token* unchanged if it finds no match.

In addition, the characters '\', '"', and '`' remove the special meaning of other characters. The backslash '\' quotes the following character. The shell ignores a backslash immediately followed by a newline, called a *concealed newline*. A pair of single quotes ' ' prevents interpretation of any enclosed special characters. A pair of double quotes " " has the same effect, except that parameter substitution and command output substitution (described below) occur within double quotes.

Scripts

Shell commands can be stored in a file, or *script*. The command

sh *script* [*parameter* ...]

executes the commands in *script* with a new subshell **sh**. Each *parameter* is an actual value for a positional parameter, as described below. If *script* has been made executable with the **chmod** command, the **sh** may be omitted.

Formal parameters of the form '$*n*', where *n* ranges from 0 through 9, represent positional parameters in a script. The parameter '$0' gives the name of the script. If no corresponding actual parameter is given on the command line, the shell substitutes the null string for the positional parameter. The shell substitutes the actual values of all positional parameters for the reference '$*'.

Commands in a script can also be executed with the . (dot) command. It resembles the **sh** command, but the current shell executes the script commands without creating a new subshell or a new environment; positional parameters are not allowed.

Variables

Shell variables are names which may be assigned string values on a command line, in the form

name = *value*

The name must begin with a letter, and may contain letters, digits, and underscores '_'. In shell input, '$*name*' or '${*name*}' represents the value of the variable. If an assignment precedes a command on the same command line, the effect of the assignment is local to the command; otherwise, the effect is permanent. For example,

```
kp=one testproc
```

assigns variable **kp** the value **one** only for the execution of the script **testproc**.

The shell sets the following variables.

The number of actual positional parameters given to the current command.

@ The list of positional parameters "$1 $2 ...".

`*` The list of positional parameters "$1" "$2" ... (the same as '$@' unless quoted).

`-` Options set in the invocation of the shell or by the **set** command.

`?` The exit status returned by the last command.

`!` The process number of the last command invoked with '&'.

`$` The process number of the current shell.

The shell also references the following variables.

HOME Initial working directory; usually specified in the password file **/etc/passwd**.

IFS Delimiters for tokens; usually space, tab and newline.

LASTERROR
Name of last command returning nonzero exit status.

MAIL Checked at the end of each command. If file specified in this variable is new since last command, the shell prints "You have mail." on the user's terminal.

PATH Colon-separated list of directories searched for commands.

PS1 First prompt string, usually '$ '.

PS2 Second prompt string, usually '> '. Used when the shell expects more input, such as when an open quote has been typed but a close quote has not been typed, or within a shell construct.

The special forms '${*namectoken*}', where *c* is one of the characters '-', '=', '+' or '?', perform conditional parameter substitution. The shell replaces the form '${*name-token*}' by the value of *name* if it is set and by *token* otherwise. The '=' form has the same effect, but also sets the value of *name* to *token* if it was not set previously. The shell replaces the '+' form by *token* if the given *name* is set. The shell replaces the '?' form by the value of *name* if set, and otherwise prints *token* and exits from the shell.

Command Output Substitution
The shell can use the output of a command as shell input (as command arguments, for example) by enclosing the command in grave accent characters ``. To list directories given in a file **dirs**, use the command

```
ls -l `cat dirs`
```

Constructs
The shell provides control over execution of commands by the **break, case, continue, for, if, until** and **while** constructs. The shell recognizes each reserved word only if it occurs unquoted as the first token of a command. This implies that a separator must precede each reserved word in the following constructs. For example, newline or ';' must precede **do** in the **for** construct.

break [*n*]
Exit from **for, until,** or **while**. If *n* is given, exit from *n* levels.

case *token* **in** [*pattern* [| *pattern*] ...) *sequence*;;] ... **esac**
Check the *token* against each *pattern*, and execute the *sequence* associated with the first matching *pattern*.

continue [*n*]
Branch to the end of the *n*th enclosing **for, until,** or **while** construct.

for *name* [**in** *token* ...] **do** *sequence* **done**
Execute *sequence* once for each member of the specified *token* list. On each iteration, the *name* takes the value of the next *token* in the list. If the **in** clause is omitted, **$@** is assumed. For example, to list all files ending with **.c**:

```
for i in *.c
do cat $i
done
```

if *seq1* **then** *seq2* [**elif** *seq3* **then** *seq4*] ... [**else** *seq5*] **fi**
Execute *seq1*. If the exit status is zero, execute *seq2*. If not, execute the optional *seq3* if given. If its exit status is zero, execute *seq4* and so on. If the exit status of each tested sequence is nonzero, execute *seq5*.

COHERENT Command

until *sequence1* [**do** *sequence2*] **done**
Execute *sequence2* until the execution of *sequence1* results in an exit status of zero.

while *sequence1* [**do** *sequence2*] **done**
Execute *sequence2* as long as the execution of *sequence1* results in an exit status of zero.

(*sequence* **)**
Execute the *sequence* within a subshell. This allows the *sequence* to change the current directory, for example, and not affect the enclosing environment.

{ *sequence* **}**
Braces simply enclose a *sequence*.

Special Commands

The shell usually executes commands by a **fork** system call, which creates another process. However, the shell executes the commands in this section either directly or with an **exec** system call. The *COHERENT System Manual* describes **fork** and **exec**.

. *script* Read and execute commands from *script*. Positional parameters are not allowed. The shell searches **PATH** to find the given *script*.

: [*token* ...]
A colon ':' indicates a comment, which the shell ignores.

cd [*dir*]
Change the working directory to *dir*. If no argument is given, change to the home directory.

eval [*token* ...]
Evaluate each *token* and treat the result as shell input.

exec [*command*]
Execute *command* directly rather than performing **fork**. This terminates the current shell.

exit [*status*]
Set the exit status to *status*, if given; otherwise, the previous status is unchanged. If the shell is not interactive, terminate it.

export [*name* ...]
The shell executes each command in an *environment*, which is essentially a set of shell variable names and corresponding string values. The shell inherits an environment when invoked, and normally it passes the same environment to each command it invokes. **export** specifies that the shell should pass the modified value of each given *name* to the environment of subsequent commands. When no *name* is given, the shell prints the name and value of each variable marked for export.

read *name* ...
Read a line from the standard input and assign each token of the input to the corresponding shell variable *name*. If the input contains fewer tokens than the *name* list, assign the null string to extra variables. If the input contains more tokens, assign the last *name* the remainder of the input.

readonly [*name* ...]
Mark each shell variable *name* as a read only variable. Subsequent assignments to read only variables will not be permitted. With no arguments, print the name and value of each read only variable.

set [**–ceiknstuvx** [*name* ...]]
Set listed flag. If *name* list is provided, set shell variables *name* ... to values of positional parameters beginning with **$1**.

shift Rename positional parameter 1 to current value of **$2**, and so on.

times Print the total user and system times for all executed processes.

trap [*command*] [*n* ...]
Execute *command* if the shell receives signal *n*. If *command* is omitted, reset traps to original values. To ignore a signal, pass null string as *command*. With *n* zero, execute *command* when the shell exits. With no arguments, print the current trap settings.

umask [*nnn*]
Set user file creation mask to *nnn*. If no argument is given, print the current file creation mask.

wait [*pid*]
Hold execution of further commands until process *pid* terminates. If *pid* is omitted, wait for all child processes. If no children are active, this command finishes immediately.

Options

−c *string*
Read shell commands from *string*.

−e Exit on any error (command not found or command returning nonzero status) if the shell is not interactive.

−i The shell is interactive, even if the terminal is not attached to it; print prompt strings. For a shell reading a script, ignore signals **SIGTERM** and **SIGINT**.

−k Place all keyword arguments into the environment. Normally, the shell places only assignments to variables preceding the command into the environment.

−n Read commands but do not execute them.

−s Read commands from the standard input and write shell output to the standard error.

−t Read and execute one command rather than the entire file.

−u If the actual value of a shell variable is blank, report an error rather than substituting the null string.

−v Print each line as it is read.

−x Print each command and its arguments as it is executed.

− Cancel the **−x** and **−v** options.

The shell reads and executes the scripts **/etc/profile** and **$HOME/.profile** before reading the standard input if the first character of argument 0 is '−'. **/etc/profile** is a convenient place for initializing system-wide variables, such as **TIMEZONE**. If the system is running in single-user mode, then the shell executes the script **/etc/.profile**.

FILES

/etc/profile	system-wide initial commands
$HOME/.profile	user-specific initial commands
/dev/null	for background input
/tmp/sh*	temporaries

SEE ALSO

login, newgrp, test
COHERENT System Manual: **dup, environ, exec, fork, signal**
sh Command Language Tutorial

DIAGNOSTICS

The shell notes on the standard error syntax errors in commands and commands which it cannot find. Syntax errors cause a noninteractive shell to exit. It gives error messages if I/O redirection is incorrect. The shell returns the exit status of the last command executed or the status specified by an **exit** command.

NAME
shift – shift positional parameters

USAGE
shift

DESCRIPTION
Commands to the shell **sh** can be stored in a file, or *script*. Positional parameters pass command line information to a script, as described in **sh**.

shift changes the values of positional parameters. The old parameter values **$2**, **$3**... become the new parameter values **$1**, **$2**... **shift** also reduces the value of '$#', which gives the number of positional parameters, by one.

The shell executes **shift** directly.

SEE ALSO
sh

NAME
size – print size of an object file

USAGE
size [**–c**] [*file* ...]

DESCRIPTION
size prints the sizes of the segments of each *file* specified in decimal bytes, and also prints the total size of all the segments in both decimal and octal bytes. Each *file* must be an object file.

If the **–c** option is specified, **size** prints the total size of all common areas in the object files as well.

One line is output for each file, listing the following segments:

shared instructions,
private instructions,
uninitialized instructions,
shared data,
private data, and
uninitialized data.

If the **–c** option is specified the total size of the common areas is displayed immediately following the uninitialized data.

SEE ALSO
COHERENT System Manual: **l.out.h**

NOTES
size makes no concessions to machines that use hexadecimal.

NAME

sleep – stop executing for a specified time

USAGE

sleep *seconds*

DESCRIPTION

sleep suspends execution for a specified number of *seconds*.

sleep is especially useful in conjunction with other commands to the shell **sh**. For example,

```
(sleep 3600; echo coffee break time) &
```

will execute the **echo** command in one hour (3600 seconds) to indicate an important appointment.

SEE ALSO

sh

COHERENT System Manual: **alarm, pause, sleep**

NAME

sort – sort lines of text

USAGE

sort [**–bcdfimnru**] [**–tc**] [**–o** *outfile*] [**–T** *dir*]
[**+***beg* [**–***end*]] [*file* ...]

DESCRIPTION

sort reads lines from each *file* specified, or the standard input if none. It writes to the standard output in sorted order. The output ordering, normally from smallest to largest, is determined by comparison of a *key* which is all or part of an input line, depending upon selected options.

By default, the key is the entire input record (line) and the ordering is the ASCII collating sequence. The following options affect how the key is constructed or how the output is ordered.

–b Ignore leading white space (blanks or tabs) in key comparisons.

–d Dictionary ordering; only letters, blanks, and digits are considered in key comparisons. This is essentially the ordering used to sort telephone directories.

–f Fold upper-case letters to lower case for comparison purposes.

–i Ignore all characters outside of the printable ASCII range (040–0176) in comparisons.

–n The key is a numeric string, consisting of optional leading blanks and minus sign followed by any number of digits with an optional decimal point. The ordering is by the numeric (as opposed to alphabetic) value of the string.

–r Reverse the ordering, i.e. sort from largest to smallest.

The key field need not be the entire input line. The option **+***beg* indicates the beginning position of the key field in the input line, and the optional **–***end* indicates that the key field ends just before the *end* position. If no **–***end* is given, the key field ends at the end of the line. Each of these positional indicators has the form **+***m.nf* or **–***m.nf*, where *m* is the number of fields to skip in the input line and *n* is the number of characters to skip after skipping fields.

Optional flags *f* are chosen from the above key flags (**bdfinr**) and are local to the specified field.

The following additional options control how **sort** works.

−c Check the input to see if it is sorted. Print the first out of order line found.

−m Merge the input files. **sort** assumes each *file* to be sorted already. For large files it runs much faster with this option.

−o *outfile*
Put the output in *outfile* rather than on the standard output. This allows **sort** to work correctly if the output file is one of the input files.

−t*c* Use the character *c* to separate fields rather than the default blanks and tabs.

−T *dir* Temporary files are created in the directory *dir* rather than the standard place.

−u Suppress multiple copies of lines with key fields which compare equally.

The following example sorts the password file **/etc/passwd**, first by group number (field 4) and then by user name (field 1):

```
sort -t: +3n -4 +0 -1 /etc/passwd
```

FILES
/usr/tmp/sort* first attempt at temporary files
/tmp/sort* second attempt at temporary files

SEE ALSO
tsort, uniq
COHERENT System Manual: **ascii, ctype**

DIAGNOSTICS
sort returns a nonzero exit status if there were file opening errors or other internal problems, or if the file was not correctly sorted in the case of the **−c** option.

NAME
spell − find spelling errors

USAGE
spell [−a] [−b] [*file* **...]**

DESCRIPTION
spell builds a set of unique words from a document contained in each input *file*, or the standard input if none. It writes a list of words believed to be misspelled onto the standard output.

spell should normally be invoked with the document in the form of the input to the text formatter **nroff** rather than the output. **spell** deletes control information to the formatter by invoking **deroff**.

The default dictionary is for American spelling of English. The **−a** option specifies this dictionary explicitly. Under the **−b** option, British spelling is checked. This accepts *favour*, *fibre*, and *travelled* rather than the American spellings *favor*, *fiber*, and *traveled* for the same words. Words ending in *ize* are also accepted when ending in *ise* (e.g. *digitize*, *digitise*).

The dictionary has a reasonably complete coverage of proper names as well as technical terms in certain fields. However, it covers some fields (e.g. computer science) better than others (e.g. medicine).

FILES
/usr/dict/clista compressed American dictionary
/usr/dict/clistb compressed British dictionary
/usr/dict/spellhist history file for dictionary maintainer
/usr/lib/spell

SEE ALSO
deroff, nroff, sort, typo

NOTES
Dictionaries are not provided for languages other than English.

No dictionary can be complete. Sites must add new words to keep users content.

Obscure words (such as opcodes, variable names, etc.) are flagged as spelling errors.

Because the data files required for **spell** are quite large, they might not be included on COHERENT systems with insufficient disk space. As a result, the command might not work as expected on all systems.

NAME
split – split a large file into smaller files

USAGE
split [*–nlines*] [*infile* [*outfile*]]

DESCRIPTION
split divides a file into a number of smaller files. This is useful for programs which cannot handle arbitrarily large files, such as **ed**.

split uses *infile* as its input file if given; otherwise, it uses the standard input. If *infile* is '–', **split** uses the standard input.

split puts its output into files with names prefixed by *outfile* and suffixed consecutively with **aa**, **ab**, **ac**, and so on. If no *outfile* is specified, file names are prefixed with **x**.

Normally, **split** puts 1000 lines in each output file. This default may be changed by the option *–nlines*, where *nlines* gives the desired number of lines per file.

SEE ALSO
ed

NAME
strip – strip symbol table and relocation information from object file

USAGE
strip *file* ...

DESCRIPTION
strip removes the symbol table and relocation information from each object *file* specified. **strip** effects reasonable savings on systems where file space is at a premium.

Since the symbol tables are used by the debugger, **strip** should not be used on an object file until it is debugged.

The **–s** option of **ld** has the same effect as **strip**.

SEE ALSO
cc, db, ld, nm

FILES
/tmp/strip* intermediate file

NAME
stty – set/print terminal modes

USAGE
stty [*option* ...]

DESCRIPTION
If no *option* is specified, **stty** prints the modes of the standard output device on the standard error stream. Otherwise each *option* modifies the modes of the standard output device. The device is usually a terminal, although tapes, disks and other special files may be applicable.

In normal processing (**cooked** mode), the *erase* and *kill* characters (normally **<ctrl-H>** and '@') erase a single typed character and a typed line, respectively. The *stop-output* and *start-output* characters (normally **<ctrl-S>** and **<ctrl-Q>**) stop and restart output. The *interrupt* character (normally **DELETE** or **RUBOUT**, ASCII 0177) sends the signal **SIGINT**, which usually terminates program execution. The *quit* character (normally ASCII 034, **FS**, which differs on various terminals but is often **<ctrl-\>**) sends the signal **SIGQUIT**, which usually terminates program execution with a core dump. The *end of file* character (normally **<ctrl-D>**) generates an end of file from the terminal. Each special character can be changed with the appropriate *option*.

On some machines, the default characters differ from those given above. On the IBM Personal Computer, for example, the default kill character is **<ctrl-U>** and the default interrupt character is **<ctrl-C>**.

The following table describes each available *option*. The *c* argument may be a literal character or may be of the form '^X' for **<ctrl-X>**.

number	Set input and output baud rates of the device to the speed *number*, if possible.
0	Hang up phone immediately.
–a	Display all modes.
break *c*	Set the break character to *c*.
cbreak	Break after every input character; allows a program to return after a read of N characters from a terminal,

even if no end of file, break or newline character was typed.

−cbreak Exit from cbreak mode.
cooked Exit from raw mode.
crt Terminal is a CRT; echoing is enhanced.
−crt The terminal is not a CRT.
echo Output characters as they are received on the input.
−echo Disable echoing.
ek Set the erase character to '#' and the kill character to '@'.
eof *c* Set the end of file character to *c*.
erase *c* Set the erase character to *c*.
even Accept even-parity characters.
−even Do not accept even-parity characters.
excl Exclusive use; subsequent opens will fail.
−excl Non-exclusive use.
flush Flush characters waiting in output or input queues.
−flush Do not flush characters.
hup Hang up the phone on last close.
−hup Do not hang up on last close.
int *c* Set the interrupt character to *c*.
kill *c* Set the kill character to *c*.
nl Disable newline mapping.
−nl Enable newline mapping; map carriage returns to linefeeds on input, and prepend carriage returns to linefeeds on output.
odd Accept odd-parity characters.
−odd Do not accept odd-parity characters.
print Print terminal attributes.
quit *c* Set the quit character to *c*.
raw Raw mode; suppress all processing and mapping (except echo).
−raw Exit from raw mode.
rawin Suppress all processing and mapping on the input stream.
−rawin Exit from rawin mode.
rawout Suppress all processing and mapping on the output stream.
−rawout Exit from rawout mode.

start *c* Set the start-output character to *c*.
stop *c* Set the stop-output character to *c*.
tabs Do not expand tabs; useful for terminals which process tabs internally.
−tabs Expand tabs to the appropriate number of spaces on output. The system assumes tabstops are at every 8th column.
tandem Tandem mode. The system will send the programmed stop-output character whenever there is a danger of losing characters from the input stream due to buffering limitations. The system will send the start-character when the level of unprocessed characters has subsided.
−tandem Disable tandem mode.

SEE ALSO
COHERENT System Manual: **ascii, getty, init, ioctl, signal**

NOTES
The system does not support character delays or mapping upper to lower case.

NAME
su – substitute user id, become superuser

USAGE
su [*user* [*command*]]

DESCRIPTION
su changes the real user id and the effective user id to that of the *user* specified. If the *user* has a login password, **su** requests it. Then it executes the specified *command*.

If *command* is absent, **su** invokes an interactive sub-shell.

If *user* is absent, **su** assumes user name **root** (the superuser).

FILES
/etc/passwd login names and passwords

SEE ALSO
login, newgrp, sh
COHERENT Administrator's Guide

DIAGNOSTICS
"Sorry" if the password is wrong.

NAME
sum – print checksum of a file

USAGE
sum [*file* ...]

DESCRIPTION
sum prints an unsigned integer checksum and a size in blocks (rounding up) for each *file* specified. If more than one *file* is specified, **sum** also prints the file name. If no *file* is specified, **sum** reads the standard input.

sum may be used to verify the integrity of data transferred across phone lines or stored on an unreliable medium (such as magnetic tape).

SEE ALSO
cmp

Maintenance

NAME
sync – flush system buffers

USAGE
sync

DESCRIPTION
Most COHERENT commands manipulate files stored on a disk. To improve system performance, the COHERENT system often changes a copy of part of the disk in a buffer in memory, rather than repeatedly performing the time-consuming disk access required.

sync writes information from the memory buffers to the disk, updating the disk images of all mounted file systems which have been changed. In addition, it writes the date and time on the root file system.

sync should be executed before system shutdown to ensure file system integrity.

SEE ALSO
COHERENT System Manual: sync

NAME
tail – print the end of a file

USAGE
tail [+*number*[bcl]] [*file*]
tail [–*number*[bcl]] [*file*]

DESCRIPTION
tail copies the last part of the specified *file*, or standard input if none, to the standard output.

The given *number* tells tail where to start copying the data. Numbers of the form +*number* measure the starting point from the beginning of the file. Numbers of the form –*number* measure from the end of the file.

A specifier of blocks, characters, or lines (b, c, or l, respectively) may follow the number. The default specification is lines.

If no *number* is specified, a default of –4l is assumed.

SEE ALSO
dd, sed

NOTES
As tail buffers data measured from the end of the file, large counts may not work.

tar

NAME

tar – tape archive manager

USAGE

tar [**crtux**[**0**–**7**]**bflmvwU**]] [*blocks*] [*archive*] *file* ...

DESCRIPTION

tar manipulates archives in a machine-independent format convenient for tape.

The first argument consists of at most one directive character, followed by zero or more option characters. The *file* arguments are generally files to be placed on or extracted from the tape. If a *file* is a directory, **tar** processes its contents recursively. For directives that input from the tape, no *file* specification tells **tar** to process every file on the tape. For directives that output to the tape, no *file* specification tells **tar** to process every file in the current directory.

The directives are:

c Create a new tape, overwriting any old contents.

r Replace (append) the named files on the tape.

t Write a table of contents of the tape to the standard output.

u Update the tape by replacing the named files which are newer (mtime larger) than any version on the tape.

x Extract the named files from the tape, overwriting existing files with the same names. **tar** extracts each version of each file, leaving the latest version at the end.

The options are:

0–**7** A single octal digit specifies the unit on which the tape may be found. **tar** concatenates this digit to the default tape name **/dev/mt** to form the pathname accessed.

b The next argument is a number between 1 and 20, specifying how many *blocks* are to be written in each tape record. **tar** determines the blocking factor automatically on input. When the blocking factor is not 1, the default tape name is **/dev/rmt** (the raw device is used).

tar

f The next argument is the name of the tape *archive*. Argument '–' means the standard input for input directives and the standard output for output directives.

l **tar** preserves links within the structure it writes to tape but breaks any links across the boundary of the structure. This option requests that **tar** report all such broken links.

m Restore the mtime for each extracted file.

v Verbose flag. If directive is **t**, the output for each file includes its mode, group id, user id, size, and mtime, in addition to its pathname. Otherwise, **tar** writes the directive and the pathname to the standard output for input directives or the standard error for output directives as each file is processed.

w For each file to be processed, **tar** writes the directive and pathname to the terminal device, then reads a line from that device and acts on it as follows:

- **n** Skip the file.
- **y** Process the file.
- **x** Exit immediately.

An empty response is treated as **n**, and end of file is treated as **x**. If a directory is skipped, all its contents are skipped. If included, all its contents are processed with this option.

U Non-COHERENT systems have another implementation of this utility with the following bug: when the blocking factor is not 1, the last few blocks of the last record written may be garbage; this bug is described elsewhere by other symptoms. This option says that the tape was created by the buggy program, so the trailing garbage should be ignored.

FILES

/dev/mt* default tape
/dev/rmt* default tape for blocking factor > 1

SEE ALSO

dump, **restor**, **tp**
COHERENT System Manual: **link**, **stat**

NOTES
Pathnames must be less than 100 characters. The m option does not affect directories. The only way to extract the Nth version of a file is with the w option.

If the m option is used to restore the mtime of an extracted file, an incremental dump may not dump the file. touch can be used to force the dump.

NAME
tee – branch pipe output

USAGE
tee [–a] [–i] [*file* ...]

DESCRIPTION
tee reads from standard input, usually a pipe, and writes to the standard output, usually a pipe. tee also writes a copy of the input data to each *file* specified.

The –a flag causes tee to append data to each *file*, analogous to the shell construct "> >*file*". Otherwise, it creates each *file*, analogous to the construct "> *file*".

The flag –i means ignore interrupts.

SEE ALSO
sh

NAME
test – evaluate conditional expression

USAGE
test *expression* ...

DESCRIPTION
test evaluates an *expression* consisting of string comparisons, numerical comparisons, and tests of file attributes. For example, a test command might be used within a shell command file to test whether a certain file exists and is readable. The logical result (true or false) of the *expression* is returned by the command, for use by a shell construct such as if.

The *expression* argument of test is constructed from the following elements, which are true if the given condition holds and false if not.

–d *file*	*file* exists and is a directory.
–f *file*	*file* exists and is not a directory.
–n *string*	*string* has nonzero length.
–r *file*	*file* exists and is readable.
–s *file*	*file* exists and has nonzero size.
–t [*fd*]	*fd* is the file descriptor number of a file which is open and a terminal. If no *fd* is given, it defaults to the standard output (file descriptor 1).
–w *file*	*file* exists and is writable.
–z *string*	*string* has zero length (is a null string).
string	*string* has nonzero length.
s1 = *s2*	string *s1* is equal to string *s2*.
s1 != *s2*	string *s1* is not equal to string *s2*.
n1 **–eq** *n2*	numbers *n1* and *n2* are equal.
n1 **–ne** *n2*	numbers *n1* and *n2* are not equal.
n1 **–gt** *n2*	number *n1* is greater than *n2*.
n1 **–ge** *n2*	number *n1* is greater than or equal to *n2*.

n1 **–lt** *n2*	number *n1* is less than *n2*.
n1 **–le** *n2*	number *n1* is less than or equal to *n2*.
! *exp*	negates the logical value of expression *exp*.
exp1 **–a** *exp2*	both expressions *exp1* and *exp2* are true.
exp1 **–o** *exp2*	either expression *exp1* or *exp2* is true. **–a** has greater precedence than **–o**.
(*exp* **)**	parentheses allow expression grouping.

The following example uses the test command to determine whether a file is writable.

```
if test ! -w /dev/lp
then
        echo the line printer is inaccessible
fi
```

On many COHERENT systems, the command '[' is linked to test. If invoked as '[', test checks that its last argument is ']'. This allows an alternative syntax: the *expression* is simply enclosed in square brackets. For example, the above example can be written as follows:

```
if [ ! -w /dev/lp ]
then
        echo the line printer is inaccessible
fi
```

SEE ALSO
expr, find, if, sh, while

NAME
time – time the execution of a command

USAGE
time [*command*]

DESCRIPTION
time invokes the given *command* with any arguments provided. Upon termination, time prints the elapsed real time, CPU time in the system, and CPU time in the user program on the standard error output.

If no *command* is given, time simply invokes date to print the current time of day.

SEE ALSO
date, prof, ps
COHERENT System Manual: times

DIAGNOSTICS
If the *command* terminates abnormally the reason is printed.

NAME
times – print total user and system times

USAGE
times

DESCRIPTION
times prints the total elapsed user time and system time for the current shell sh and all its children. It gives each time in minutes, seconds and tenths of seconds. For example,

```
1m11.8s 1m35.8s
```

indicates a total user time of 1 minute 11.8 seconds, and a total system time of 1 minute 35.8 seconds.

The shell executes times directly.

SEE ALSO
sh

NAME
touch – update modification time of a file

USAGE
touch [–c] *file* ...

DESCRIPTION
The COHERENT system keeps track of when each file was last modified. **touch** changes the modification time of each *file* to the current time, but does not modify the contents of the file. If the **–c** flag is present, **touch** will not create *file* if it does not already exist.

SEE ALSO
make

NOTES
touch works by reading a character from each file and writing it back, so it fails on device files.

NAME
tp – maintain files on tape

USAGE
tp [dprtux[0–7cfhilmvw] ...] [*name* ... **]**

DESCRIPTION
tp maintains a tape consisting of an unused block, 62 blocks of directory, and blocks containing the file data. Files begin on block boundaries.

The first argument consists of at most one directive character and zero or more option characters. If omitted, it defaults to **um**. The *name* arguments are generally files to be placed on or extracted from the tape. If a *name* is a directory, **tp** processes its contents recursively. For directives that input from the tape, no *name* argument tells **tp** to process every file on the tape; for directives that output to the tape, no *name* tells **tp** to process '.' (the current directory). **tp** compares file names on the command line and on the tape naively; **file1** is not the same as **./file1**.

The directives are:

d Delete the named files from the tape.

p Print the named files on the standard output.

r Replace the named files on the tape; new files are appended.

t Write a table of contents of the tape to the standard output.

u Update the tape by replacing the named files which are newer (mtime larger) than the copy on the tape.

x Extract the named files from the tape, overwriting existing files with the same names.

The options are:

0–7 A single octal digit specifies the unit on which the tape is found. **tp** concatenates this digit with the default tape name to form the pathname accessed.

c Create the tape, i.e. overwrite from the beginning. This option is implied on output to magtape, since magtape may

not be selectively overwritten. This means **u** is equivalent to **r** on magtape.

f The next argument is the tape name. The name '–' means the standard input for input directives and the standard output for output directives; **c** must be supplied or implied.

h The tape is a high density (1600 bpi) magtape. **tp** uses this information in length calculations only.

i After detecting and reporting an I/O error on the tape, continue execution instead of exiting.

l The next argument is a number specfying the length of the tape in feet (default: 2400). **tp** is careful not to exceed this limit.

m The medium is magtape (by default at **/dev/mt**).

v Verbose flag. If the directive is **t**, the output for each file includes mode, group id, user id, size, block address (in **tp** file), and mtime, in addition to the pathname. For other directives, write the directive and the pathname to the standard error as each file is processed. This overrides **w** (see below).

w For each file, write the directive and pathname to the standard error. Then read a line from the terminal **/dev/tty**, and act on it as follows:

- **n** Skip the file.
- **y** Process the file.
- **x** Exit immediately.

tp treats an empty response as **n** and treats end of file as **x**.

FILES
/dev/mt default magtape
/dev/tty for w option

SEE ALSO
tar
COHERENT System Manual: **stat**

DIAGNOSTICS
tp diagnostics are self-explanatory.

NOTES
DECtape is not supported. Restrictions: file names must be less than 32 characters; there cannot be more than 496 files; a magtape cannot be updated.

NAME
tr – translate characters

USAGE
tr [–cds] *string1* [*string2*]

DESCRIPTION
tr reads characters from the standard input, possibly translates each to another value or deletes it, and writes to standard output. A character is an 8-bit ASCII value in the range \0 – \377.

Each specified *string* may contain literal characters of the form *a* or *b* (where *b* is non-numeric), octal representations of the form *ooo* (where *o* is an octal digit), and character ranges of the form "*X* – *Y*". **tr** rewrites each *string* with the appropriate conversions and range expansions.

If an input character is in *string1*, **tr** outputs the corresponding character of *string2*. If *string2* is shorter than *string1*, the result is the last character in *string2*.

If the complement flag –c is present, **tr** replaces *string1* by the set of characters not in *string1*.

If the deletion flag –d is present, **tr** deletes characters in *string1* rather than translating them.

If the squeeze option –s is present, **tr** maps a sequence of the same character from *string1* in the input to a single output character.

For example,

```
tr -cs ' -~' '\12' <infile | grep ....
```

prints all sequences of four or more spaces or printing characters from **infile**. Here *string1* is the range from ' ' to '~', which includes all printing characters. Since the flags –cs are used, **tr** maps sequences of nonprinting characters to newline (octal 12).

SEE ALSO
sed
COHERENT System Manual: **ascii, ctype**

NAME
trap – execute command on receipt of signal

USAGE
trap [*command*] [*n* ...]

DESCRIPTION
trap instructs the current shell **sh** to execute the given *command* when the shell receives signal *n* or any other signal in the optional list. If the *command* is omitted, **trap** resets traps for the given signals to the original values. If the *command* is the null string "", the shell ignores the given signals. If *n* is 0, the shell executes the specified *command* when it exits. With no arguments, **trap** prints the signal number and command for each signal on which a trap is set.

The shell executes **trap** directly.

SEE ALSO
sh
COHERENT System Manual: **signal**

NAME
true – unconditional success

USAGE
true

DESCRIPTION
true simply returns success and has no other side effects. It can be useful in shell scripts:

```
while true; do
        date
done
```

DIAGNOSTICS
Exit status is 0.

NAME
tsort – topological sort

USAGE
tsort [*file*]

DESCRIPTION
tsort performs a topological sort of a set of input items. The input *file* (or the standard input, if no *file* is given) specifies an ordering on pairs of items. It consists of pairs of items separated by blanks, tabs or newlines. If a pair contains the same item twice, it simply indicates that the item is in the input set. Otherwise, the pair indicates that the first item precedes the second in the ordering.

tsort prints a sorted list of the input items on the standard output.

SEE ALSO
sort

DIAGNOSTICS
tsort prints an error message on the standard error if its input contains an odd number of items or if the specified ordering includes a cycle.

NAME
tty – print the user's terminal name

USAGE
tty

DESCRIPTION
tty prints the name of the character special file which is the user's terminal.

DIAGNOSTICS
tty prints the message "Not a tty." if the user is not associated with any controlling terminal.

SEE ALSO
who

NAME
typo – detect possible typographical and spelling errors

USAGE
typo [–nrs] [*file* ...]

DESCRIPTION
typo proofreads an English language document for typographical errors. It conducts a statistical test of letter digrams and trigrams in each input word against digram and trigram frequencies throughout the entire document. From this test, typo computes an index of peculiarity for each word in the document; a high index indicates a word less like other words in the document than a low index. Built-in frequency tables ensure reasonable results even for relatively short documents.

typo reads each input *file* (or the standard input if none), and removes punctuation and non-alphabetic characters to produce a list of the words in the document. To reduce the volume of the output, typo compares each word against a small dictionary of technical words and discards it if found. The output consists of a list of unique non-dictionary words with associated index of peculiarity, most peculiar first. An index higher than 10 indicates that the word almost certainly occurs only once in the document.

Options:

–n inhibits use of the built-in English digram and trigram statistics, and inhibits dictionary screening of words. More words will be output and the indices of peculiarity will be less useful for short documents.

–r inhibits the default stripping of **nroff** escape sequences. Normally typo strips lines beginning with '.' and removes the **nroff** escape sequences '\'.

–s instructs typo to produce output files **digrams** and **trigrams** containing digram and trigram frequency statistics for the given document. No indices of peculiarity are calculated or printed. If desired, these files may be installed in directory **/usr/dict**.

FILES

/tmp/typo*	intermediate files
/usr/dict/dict	limited dictionary
/usr/dict/digrams	digram frequency statistics
/usr/dict/trigrams	trigram frequency statistics

SEE ALSO
nroff, sort, spell

NOTES
Because the data files required for **typo** are quite large, they might not be included on COHERENT systems with insufficient disk space. As a result, the command might not work as expected on all systems.

Maintenance

NAME
uload – unload device driver

USAGE
/etc/uload *file*

DESCRIPTION
uload unloads the device driver *file*. The argument should be the same as previously given to the **load** command. By convention, device drivers are stored in the **/drv** directory.

uload is restricted to the superuser.

FILES
/drv/* driver files

SEE ALSO
load
COHERENT System Manual: **sload**

NOTES
Because of hardware restrictions, the COHERENT system does not support loadable device drivers on systems based on the 8086 or 8088 processors (such as the IBM Personal Computer). The **/drv** directory and the **uload** command do not exist on such systems.

NAME
umask – set/print file creation mask

USAGE
umask [*nnn*]

DESCRIPTION
The COHERENT command interpreter **sh**, called the shell, keeps track of a *file creation mask*. Whenever the shell creates a new file, its protection bits are taken to be the logical AND of the specified mode and the complement of the file creation mask. Bits set in the mask specify which protections will be denied automatically when a file is created.

umask sets or prints the file creation mask. If an octal argument *nnn* is given, it becomes the new value of the mask. Otherwise, **umask** prints the current value of the mask. The shell executes **umask** directly.

For example, the command

```
umask 077
```

will deny read, write and execute permission to all other users for each file created subsequently.

SEE ALSO
chmod, sh
COHERENT System Manual: **chmod, creat, umask**

Maintenance

NAME
umount – unmount file system

USAGE
/etc/umount *special*

DESCRIPTION
umount unmounts a file system *special* which was previously mounted by the **mount** command.

FILES
/etc/mtab mount table
/dev/*

SEE ALSO
check, clri, icheck, mount
COHERENT System Manual: **mount**

DIAGNOSTICS
Errors can occur if *special* does not exist or is not a mounted file system.

NAME
uniq – remove/count repeated lines in a sorted file

USAGE
uniq [**–cdu**] [*–n*] [*+n*] [*infile* [*outfile*]]

DESCRIPTION
uniq normally reads input line by line from *infile* and writes all non-repeated lines to *outfile*. The input must be sorted. **uniq** uses the standard input and output if either *infile* or *outfile* is omitted.

The option **–u** prints only non-repeated lines; option **–d** prints each duplicated line once. Normally **uniq** behaves as if both **–u** and **–d** were specified.

If **–c** is specified, **–u** and **–d** are superseded, and a count of the repetitions precedes each output line.

Optional specifiers allow skipping leading portions of the input lines when comparing for uniqueness.

–n skips *n* fields of each input line, where a field is any number of non-white space characters surrounded by any number of white space characters (blank or tab).

+n skips *n* characters in each input line, after skipping fields as above.

SEE ALSO
comm, sort

NAME
units – measurement conversion

USAGE
units [**–u**]

DESCRIPTION
units is an interactive program for converting one unit of measurement to another. It asks for two quantities with the same dimensions and prints the scale factors to get from one to the other. For example:

```
You have: fortnight
You want: days
* 14
/ 0.071428
```

The fundamental units are: **meter, gram, second, coulomb, radian, bit, unitedstatesdollar, sheet, candle, kelvin,** and **copperpiece** (D&D).

A quantity consists of an optional number (default 1) and a dimension (default none). Numbers are floating point with optional sign, decimal part and exponent. Dimensions may be specified by fundamental or derived units, with optional orders. A quantity is evaluated left to right: a factor preceded by a '/' is a divisor, otherwise it is a multiplier. For example, Earth's gravitational acceleration may be entered as any of the following:

```
9.8e+0 m+1 sec-2
32 ft/sec/sec
32 ft/sec+2
```

British equivalents of US units are prefixed with **br**, e.g. **brpint**. Some other units are: **c** (speed of light), **G** (gravitational constant), **R** (gas law constant), **phi** (golden ratio), **%** (1/100), **k** (1024) and **buck** (unitedstatesdollar).

/usr/lib/units is an ASCII file containing conversion tables. The binary file **/usr/lib/binunits** may be recreated by using the **–u** option.

SEE ALSO
bc, conv

FILES
/usr/lib/units known units
/usr/lib/binunits binary encoding of units file

DIAGNOSTICS
If the ASCII file **/usr/lib/units** has been changed more recently than the binary file **/usr/lib/binunits**, **units** prints "Waiting for Godot..." and regenerates the binary file before continuing; this takes at least 90 seconds. "conformability" means that the quantities are not dimensionally compatible, for example, **m/sec** and **psi**. **units** prints each quantity and its dimensions in fundamental units.

NOTES
There are the inevitable name collisions: **g** for gram vs. **gee** for Earth's gravitational acceleration, **exp** for the base of natural logarithms vs. **e** for the charge of an electron, **ms** for (plural) meters vs. **millisecond**, and of course **batman** for the Persian measure of weight rather than the Turkish.

NAME
until – execute commands repeatedly

USAGE
until *sequence1*
[do *sequence2* **]**
done

DESCRIPTION
The shell **until** loop construct first executes the commands in *sequence1*. If the exit status is nonzero, the shell executes the commands in the optional *sequence2* and repeats the process until the exit status of *sequence1* is zero. Because the shell recognizes a reserved word only as the unquoted first word of a command, both **do** and **done** must occur unquoted at the start of a line or preceded by ';'.

The shell commands **break** and **continue** may be used to alter control flow within an **until** loop. The **while** construct has the same form as **until** but the sense of the test is reversed.

The shell executes **until** directly.

SEE ALSO
break, continue, sh, test, while

NAME
wait – await completion of background process

USAGE
wait [*pid*]

DESCRIPTION
Typing the character '&' after a command tells the shell **sh** to execute it as a *background* (or *detached*) process; otherwise, it is executed as a *foreground* process. A user can proceed with other tasks while a background process is being performed. The shell prints the process id number of each background process when it is invoked. **ps** reports on currently active processes.

wait suspends execution until the child process with the given *pid* is completed. If no *pid* is given, **wait** suspends execution until all background processes are completed. If the process with the given *pid* is not a child process of the current shell, **wait** returns immediately.

The shell executes **wait** directly.

SEE ALSO
ps, sh

NOTES
If a subshell invokes a background process and then terminates, **wait** will return immediately rather than waiting for termination of the grandchild process.

NAME
wall – send a message to all logged in users

USAGE
/etc/wall

DESCRIPTION
wall (write all users) types a message to every user (except the sender) currently logged into the COHERENT system. It informs users of information of general interest; for example, that man has landed on the moon, or that the system is going down for backup in fifteen minutes.

wall reads the message to be broadcast from the standard input until end of file. When it sends the message, it prefaces it with the herald "Broadcast message ...", which includes an audible warning. Only the superuser should invoke **/etc/wall** (to override access protections of the target terminals).

SEE ALSO
msg, who, write
COHERENT Administrator's Guide

DIAGNOSTICS
The message "Cannot send to *user* on *ttyname*" indicates inability to write to the given *user*.

FILES
/etc/utmp current users file
/dev/tty*

NAME
wc – count words, lines, and characters in files

USAGE
wc [–clw] [*file* **...]**

DESCRIPTION
wc counts words, lines, and characters in each *file* named. If no *file* is given, wc uses the standard input. If more than one *file* is given, wc also prints a total.

A *word* is a string of characters surrounded by white space (blanks, tabs, or newlines).

Options control the printing of various counts:

–c print character count

–l print line count

–w print word count

The default action is to print all counts.

NAME
while – execute commands repeatedly

USAGE
while *sequence1*
[do *sequence2* **]**
done

DESCRIPTION
The shell **while** loop construct first executes the commands in *sequence1*. If the exit status is 0, the shell executes the commands in the optional *sequence2* and repeats the process until the exit status of *sequence1* is nonzero. Because the shell recognizes a reserved word only as the unquoted first word of a command, both **do** and **done** must occur unquoted at the start of a line or preceded by ';'.

The shell commands **break** and **continue** may be used to alter control flow within a **while** loop. The **until** construct has the same form as **while** but the sense of the test is reversed.

The shell executes **while** directly.

SEE ALSO
break, continue, sh, test, until

NAME
who – print who is logged in

USAGE
who [*file*] [**am i**]

DESCRIPTION
With no arguments, **who** lists the users currently logged in to the COHERENT system. For each user, **who** prints the username, terminal name, login date, and login time.

The form **who am i** prints only the information about the user.

If *file* is specified, **who** uses it instead of **/etc/utmp** to obtain the information. This is useful, for example, with the file **/usr/adm/wtmp**, which contains a continuous record of logins, logouts and reboots. When *file* is specified, **who** displays logouts; otherwise, they are suppressed.

FILES
/etc/utmp to get user information

SEE ALSO
ac, sa

NAME
write – conduct interactive conversation with another user

USAGE
write *user* [*tty*]

DESCRIPTION
The COHERENT system provides several commands which facilitate communication between users. The **write** command allows two currently logged in users to engage in an extended interactive conversation. The **msg** command should be used to send brief communications to a logged in user, and the **mail** command should be used to communicate with a user not currently logged in. The **wall** command broadcasts a message to all logged in users.

write initiates a conversation with the *user* specified. If *tty* is given, **write** looks for the *user* on that terminal; otherwise, **write** holds the conversation with the first instance of *user* found on any tty.

If found, the specified *user* receives notification that you are beginning a conversation with him. All subsequent lines typed into **write** are forwarded to the *user*'s terminal, except that lines beginning with '!' are sent to the shell **sh**. Typing end of file (usually <**ctrl-D**>) terminates **write** and notifies the *user* that communication is terminated with the message "EOT".

Two users typing lines to **write** at about the same time can cause extreme confusion, so users should agree on a protocol to limit when each is typing. The following protocol is suggested. One user initiates a **write** to another, and waits until the other user replies before beginning. The first user then types until he wishes a reply and suffixes "**o**" (over) to indicate he is through. The other user does the same, and the conversation alternates until one user wishes to terminate it. This user types "**oo**" (over and out). The other user replies in the same way, indicating he too is finished. Finally each user leaves **write** by typing end of file.

Any user may deny others the permission to **write** to his terminal by means of the **mesg** command.

FILES
/etc/utmp
/dev/*

SEE ALSO
mail, mesg, msg, sh, wall, who

NAME
xdecode – decode private message

USAGE
xdecode

DESCRIPTION
xdecode decodes a private message using a public key cryptosystem, described in detail in **enroll**. **xdecode** prompts for a passphrase to use in decryption. Then it decrypts the standard input and writes the result on the standard output.

enroll enrolls a user in the public key cryptosystem. **xencode** encodes a private message. **xmail** sends or reads mail through the cryptosystem.

FILES
/usr/spool/pubkey/*user* for *user*'s public key

SEE ALSO
crypt, enroll, xdecode, xmail

NAME
xencode – encode private message

USAGE
xencode [*user*]

DESCRIPTION
xencode encodes a private message using a public key cryptosystem, described in detail in **enroll**. If a *user* is given, **xencode** uses *user*'s public key for the encryption; otherwise, it prompts for a passphrase and generates a public key from it. Then it encrypts the standard input and writes the result on the standard output.

enroll enrolls a user in the public key cryptosystem. **xdecode** decodes a private message. **xmail** sends or reads mail through the cryptosystem.

FILES
/usr/spool/pubkey/*user* for *user*'s public key

SEE ALSO
crypt, enroll, xencode, xmail

NAME
xmail – secret computer mail

USAGE
xmail *user* ...
xmail [**–nxy**]

DESCRIPTION
xmail sends or reads secret mail through a public key cryptosystem, described in detail in **enroll**. Recipients of secret mail must be enrolled in the system, but the sender need not be enrolled.

If any *user* is specified, **xmail** reads a message from the standard input and sends it in encoded form to each *user*. Either the end of file character (normally <ctrl-D>) or a line containing '.' terminates the message. **xmail** stores the encoded mail in the secret mail file **xmailbox** in the *user*'s home directory and sends *user* a message via **mail** indicating that the user has received secret mail.

If no *user* is given, **xmail** decodes secret mail. It prompts for a passphrase (see **enroll**) and writes decrypted mail on the standard output. Options control the action taken with the secret mail. Option **–n** throws the mail away after decoding. **–x** saves the decoded secret mail in file **mbox** in the user's home directory, while **–y** saves the mail in encrypted form in file **xmbox** in the user's home directory. If the options are missing, **xmail** asks what action it should take. The replies **n**, **x** and **y** cause the above actions; any other reply causes no action.

To read mail stored in encrypted form with the **–y** option, use **xdecode**:

```
xdecode <$HOME/xmbox
```

FILES
$HOME/mbox	saved decoded mail
$HOME/xmailbox	secret mailbox
$HOME/xmbox	saved encoded mail
/usr/spool/mail	regular mailbox directory
/usr/spool/pubkey	public key directory

SEE ALSO
crypt, enroll, mail, xdecode

NOTES
xmail should be fully integrated with regular **mail**.

NAME
yacc – parser generator

USAGE
yacc [*option* ...] *file*
cc y.tab.c [**–ly**]

DESCRIPTION
Many programs process highly structured input according to given rules; compilers are a familiar example. Two of the most complicated parts of such programs are *lexical analysis* and *parsing* (sometimes called *syntax analysis*). The COHERENT system includes two powerful tools called **lex** and **yacc** to assist the programmer. **lex** converts a set of lexical rules to a lexical analyzer, and **yacc** converts a set of parsing rules to a parser. The output of **lex** can be used directly, or can be used by a parser generated by **yacc**.

The *yacc Parser Generator Tutorial* describes **yacc** in detail. In brief, the **yacc** ("yet another compiler-compiler") input *file* describes a context free grammar using a BNF-like syntax. The output is a file **y.tab.c** containing the definition of a C function **yyparse()** which parses the language described in *file*. The output is ready for processing by the C compiler **cc**. Ambiguities in the grammar are reported to the user, but resolved automatically by precedence rules. The user must provide a lexical scanner **yylex()**, for example by using **lex**. The **yacc** library includes default definitions of **main**, **yylex** and **yyerror**, and may be included with the option **–ly** on the **cc** command line.

Options:

–d Enable debugging output; implies **–v**.

–hdr *headerfile*
Put the header output in *headerfile* instead of **y.tab.h**.

–l *listfile* Place a description of the state machine, tokens, parsing actions and statistics in file *listfile*.

–st Print statistics on the standard output.

–v Verbose. Like **–l**, but places the listing in file **y.output** by default.

The following options are useful if table overflow messages appear:

–nterms *N* Allow for *N* nonterminals; default 100.

–prods *N* Allow for *N* productions (rules); default 175.

–states *N* Allow for *N* states; default 300.

–terms *N* Allow for *N* terminal symbols; default 100.

–types *N* Allow for *N* types; default 10.

FILES

y.tab.c	C source output
y.tab.h	default C header output
y.output	default listing output
/lib/yyparse.c	protoparser
/tmp/y[ao]*	temporaries
/usr/include/action.h	header referenced by protoparser
/usr/lib/liby.a	library

SEE ALSO

cc, lex

yacc Parser Generator Tutorial

Frank DeRemer and Thomas J. Pennello, "Efficient Computation of LALR(1) Lookahead Sets", *SIGPLAN Conference, 1979.*

DIAGNOSTICS

yacc reports the number of R/R (reduce/reduce) and S/R (shift/reduce) conflicts (ambiguities) on the standard error stream.

NAME

yes – print infinitely many responses

USAGE

yes [*answer*]

DESCRIPTION

With no argument, **yes** prints an arbitrary number of lines consisting solely of 'y'. If the optional *answer* is given, then **yes** produces lines containing the *answer* instead. For example,

```
yes n
```

is infinite refusal.

To keep this manual and COHERENT free of bugs and facilitate future improvements, we would appreciate receiving your reactions. Please fill in the appropriate sections below and mail to us. Thank you.

Mark Williams Company
1430 W. Wrightwood Avenue
Chicago, IL 60614

Name: _____

Company: _____

Address: _____

_____

Phone: _____ Date: _____

Version and hardware used: _____

Did you find any errors in the manual? _____

_____

_____

Can you suggest any improvements to the manual? _____

_____

_____

Did you find any bugs in the software? _____

_____

_____

Can you suggest improvements or enhancements to the software?

_____

_____

_____

Additional comments: (Please use other side.)