c — Command

Print multi-column output
c [-l*N* **] [-w***N* **] [-012]**

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.

c recognizes the following options:

-l*N* Set the length of the page to *N* lines. **c** columnizes its output by pages when this option is used with mode 1 or mode 2.

-w*N* Set the width of the page to *N* characters. The 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 variable **C**, separated by white space. Command line options override options in the environment. For example,

```
export C="-l56 -w72 -2"
c -w80 <file1
```

has the same effect as

```
c -l56 -w72 -2 -w80 <file1
```

This command sets the page width to 80 rather than to 72.

See Also

commands, export, pr

Diagnostics

c prints "out of memory" and returns an exit status of one if it cannot allocate enough memory to process its input.

C keywords — Overview

A **keyword** is a word that is reserved within C, and must not be used to name variables, functions, or macros. COHERENT recognizes the following C keywords:

alien	**auto**	**break**
case	**char**	**const**
continue	**default**	**do**
double	**else**	**enum**
extern	**float**	**for**
goto	**if**	**int**
long	**register**	**return**
short	**signed**	**sizeof**
static	**struct**	**switch**
typedef	**union**	**unsigned**
void	**volatile**	**while**

In conformity with the ANSI standard, the keywords **entry** and **readonly** are no longer recognized. The ANSI Standard transfers the functionality for **readonly** to the keyword **const**. For details, see the Lexicon entry for **const**.

The COHERENT C compiler recognizes the keywords **const** and **volatile**, but ignores them.

The following tokens are C++ keywords:

class
inline
private
protected
public

Your programs should avoid using them in the interest of compatibility with future versions of the COHERENT C compiler.

See Also

C language

C language — Overview

COHERENT includes a C compiler that fully implements the Kernighan and Ritchie standard of C, with extensions taken from the ANSI standard.

Please note that in the following discussion, *word* indicates an object 16 bits long; *dword*, an object 32 bits long; and *qword*, an object 64 bits long:

Identifiers

Characters allowed: **A-Z**, **a-z**, **_**, **0-9**
Case sensitive
Number of significant characters in a variable name: **255**

Escape Sequences

The COHERENT C compiler recognizes the following escape sequences:

	ASCII	*Ctrl*	*Hex*	*Description*
\a	BEL	<ctrl-G>	0x07	audible tone (bell)
\b	BS	<ctrl-H>	0x08	backspace
\f	FF	<ctrl-L>	0x12	formfeed
\n	LF	<ctrl-J>	0x0A	linefeed (newline)
\r	CR	<ctrl-M>	0x0D	carriage return
\t	HT	<ctrl-I>	0x09	horizontal tab
\v	VT	<ctrl-K>	0x0B	vertical tab
\xhh			0xhh	hex (one to four hex digits [0-9a-fA-F])
\ooo				octal (one to four octal digits [0-7])

Trigraphs

The COHERENT C compiler recognizes the following trigraphs:

Trigraph Sequence	*Character Represented*
??=	#
??(	[
??/	\
??)	]
??'	^
??<	{
??!	|
??>	}
??-	~

For details, see the Lexicon entry **trigraph**.

Reserved Identifiers (Keywords)

See the Lexicon entry for **C keywords**.

Data Formats (in bits)

char	8
unsigned char	8
double	64
enum	8|32
float	32
int	32
unsigned int	32
long	32
unsigned long	32
pointer	32
short	16
unsigned short	16

Floating-Point Formats

IEEE floating-point **float**:
- 1 sign bit
- 8-bit exponent
- 24-bit normalized fraction with hidden bit
- Bias of 127

IEEE floating-point **double**:
- 1 sign bit
- 11-bit exponent
- 53-bit fraction
- Bias of 1,023

Reserved values:
- +- infinity, -0

All floating-point operations are done as **double**s.

Limits

Maximum bitfield size: 32 bits
Maximum number of **case**s in a **switch**: no formal limit
Maximum number of arguments in function declaration: 32
Maximum number of arguments in function call: no formal limit
Maximum block nesting depth: no formal limit
Maximum parentheses nesting depth: no formal limit
Maximum structure size: no formal limit
Maximum array size: no formal limit

Preprocessor Instructions

#define **#ifdef**
#else **#ifndef**
#elif **#include**
#endif **#line**
#if **#undef**
#pragma

Structure Name-Spaces
Supports both Berkeley and Kernighan-Ritchie conventions for structure in union.

Function Linkage
Return values in EAX
Return values for **double**s:
With software floating-point emulation returns in EDX:EAX
Hardware floating-point (-VNDP) returns in the NDP stacktop **%st0**
Parameters pushed on stack in reverse order:
chars, **short**s, and pointers pushed as dwords
Structures copied onto the stack
Caller must clear parameters off stack
Stack frame linkage is done through ESP register

Structures and Alignment
Structure members are aligned according to the most strictly aligned type within the structure. For example, a structure is word-aligned if it contains only **short**s, but on dword if it contains an **int** or **long**. **#pragma align** *n* can override this feature.

Registers
Registers EBX, EDI, and ESI are available for register variables. Only 32-bit objects go into registers.

Special Features and Optimizations

Both implementations of C perform the following optimizations:

- Branch optimization is performed: this uses the smallest branch instruction for the required range.
- Unreached code is eliminated.
- Duplicate instruction sequences are removed.
- Jumps to jumps are eliminated.
- Multiplication and division by constant powers of two are changed to shifts when the results are the same.
- Sequences that can be resolved at compile time are identified and resolved.

Compilation Environments

COHERENT supports a number of different compilation environments. For example, you can compile a program to use the environment for UNIX System V release 4 or release 3, or the Berkeley environment. This is done by setting manifest constants on your C compiler's command line, which, in turn, invokes various settings within the header files. For details, see the Lexicon entry for **header files**.

Example

The following gives an example C program, which does something interesting. It was writen by Charles Fiterman:

```
char *x="char *x=%c%s%c;%cmain(){printf(x,34,x,34,10,10);}%c";
main(){printf(x,34,x,34,10,10);}
```

See Also

argc, argv, C keywords, C preprocessor, environ, envp, header files, initialization, libraries, main(), name space, offsetof(), Programming COHERENT, trigraph

C preprocessor — Overview

Preprocessing encompasses all tasks that logically precede the translation of a program. The preprocessor processes headers, expands macros, and conditionally includes or excludes source code.

Directives

The C preprocessor recognizes the following directives:

#if . . . . . Include code if a condition is true
#elif . . . . . Include code if directive is true
#else . . . . . Include code if preceding directives fail
#endif . . . . . End of code to be included conditionally

#ifdef . . . . . Include code if a given macro is defined
#ifndef . . . . . Include code if a given macro is not defined

#define . . . . . Define a macro
#undef . . . . . Undefine a macro
#include . . . . . Read another file and include it
#line . . . . . Reset current line number

The COHERENT preprocessor also recognizes the directive **#pragma**, which performs implementation-specific tasks. See the Lexicon entry on **#pragma** for details.

A preprocessing directive is always introduced by the '#' character. The '#' must be the first non-white space character on a line, but it may be preceded by white space and it may be separated from the directive name that follows it by one or more white space characters.

Preprocessing Operators

The Standard defines two operators that are recognized by the preprocessor: the "stringize" operator **#**, and the "token-paste" operator **##**. It also defines a new keyword associated with preprocessor statements: **defined**.

The operator **#** indicates that the following argument is to be replaced by a string literal; this literal names the preprocessing token that replaces the argument. For example, consider the macro:

```
#define display(x) show((long)(x), #x)
```

When the preprocessor reads the line

```
display(abs(-5));
```

it replaces it with the following:

```
show((long)(abs(-5)), "abs(-5)");
```

The **##** operator performs "token pasting" — that is, it joins two tokens together, to create a single token. For example, consider the macro:

```
#define printvar(x) printf("%d\n", variable ## x)
```

When the preprocessor reads the line

```
printvar(3);
```

it translates it into:

```
printf("%d\n", variable3);
```

In the past, token pasting had been performed by inserting a comment between the tokens to be pasted. This no longer works.

Predefined Macros

The ANSI Standard describes the following macros that must be recognized by the preprocessor:

__DATE__ Date of translation
__FILE__ Source-file name
__LINE__ Current line within source file
__STDC__ Conforming translator and level
__TIME__ Time of translation

For more information on any one of these macros, see its entry.

Conditional Inclusion

The preprocessor will conditionally include lines of code within a program. The directives that include code conditionally are defined in such a way that you can construct a chain of inclusion directives to include exactly the

material you want.

The preprocessor keyword **defined** determines whether a symbol is defined to the **#if** preprocessor directive. For example,

```
#if defined(SYMBOL)
```

or

```
#if defined SYMBOL
```

is equivalent to

```
#ifdef SYMBOL
```

except that it can be used in more complex expressions, such as

```
#if defined FOO && defined BAR && FOO==10
```

defined is recognized only in lines beginning with **#if** or **#elif**.

Note that **defined** is a preprocessor keyword, not a preprocessor directive or a C keyword. You could, for example, write a function called **defined()** without any complaint from the C compiler.

The COHERENT preprocessor implicitly defines the following macros:

```
__COHERENT__
__MWC__
__IEEE__
__I386__

_IEEE
_I386
MWC
COHERENT
```

These can be used to include conditionally code that applies to a specific edition of COHERENT. COHERENT 286 uses DECVAX floating-point code; whereas COHERENT 386 uses IEEE. If you were writing code that intensively used floating-point numbers and you wanted to compile the code under both editions of COHERENT, you could write code of the form:

```
#ifdef _DECVAX
      ...
#elif _IEEE
      ...
#endif
```

The C preprocessor under each edition of COHERENT would ensure that the correct code was included for compilation.

Macro Definition and Replacement

The preprocessor performs simple types of macro replacement. To define a macro, use the preprocessor directive **#define** *identifier value*. The preprocessor scans the translation unit for preprocessor tokens that match *identifier*; when one is found, the preprocessor substitutes *value* for it.

Inclusion of Macros or Functions

The ANSI standard demands that every routine implemented as a macro also be implemented as a function, with the exception of the macro **va_arg()**. For example, COHERENT implements the STDIO routines **toupper()** and **tolower()** both as macros and functions.

By default, COHERENT uses the macro version of routines. To force it to use the function of a routine, you must undefine the macro version. You can do that either by using the preprocessor instruction **#undef** in your code, or by using the option **-U** on the **cc** command line. For example, to compel COHERENT to use the function version of **tolower()**, include the statement

```
#undef tolower
```

in your program, or include the argument

```
-Utolower
```

on the **cc** command line.

cpp

Under COHERENT, C preprocessing is done by the program **cpp**. The **cc** command runs **cpp** as the first step in compiling a C program. **cpp** can also be run by itself.

cpp reads each input *file*; it processes directives, and writes its product on **stdout**.

If its **-E** option is not used, **cpp** also writes into its output statements of the form **#line** *n filename*, so that the parser **cc0** can connect its error messages and debugger output with the original line numbers in your source files.

See the Lexicon entry on **cpp** for more information.

See Also

C language, cc, cpp, defined, macro, manifest constant,

cabs() — Mathematics Function (libm)

Complex absolute value function
#include <math.h>
double cabs(*z***) struct { double** *r***,** *i***; }** *z***;**

cabs() computes the absolute value, or modulus, of its complex argument *z*. The absolute value of a complex number is the length of the hypotenuse of a right triangle whose sides are given by the real part *r* and the imaginary part *i*. The result is the square root of the sum of the squares of the parts.

Example

For an example of this function, see the entry for **acos()**.

See Also

hypot(), lib

cal — Command

Print a calendar
cal [*month*] [*year*]

cal prints a calendar for the specified *year* (by default, the current year), or for the given *month* if one is specified. If neither is specified, a calendar of the current month is printed. *year* must be between 1 and 9999. *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
```

See Also

commands

Notes

cal assumes that the Gregorian calendar was adopted on September 3, 1752, which is the date of its adoption throughout the British empire.

calendar — Command

Reminder service
calendar [-a] [-f*file* **]... [-d[***date***]] [-w[***date***]] [-m[***month***]]**

calendar is the COHERENT system's "reminder service". It reads a calendar file, which should contain information organized by date; if an event is scheduled to happen today or tomorrow, **calendar** prints the entry on the standard output. Thus, you can use **calendar** to remind you of both one-time events (such as appointments) and yearly events (such as anniversaries).

calendar recognizes the following command-line options:

-a Search the calendars of all users and send mail. Default is to search only your calendar.

-f*file* Search each "file" in order given. Default is **$HOME/.calendar**.

-d[*date***]** Print all entries for "date". Default date is today.

-w[*date***]** Print all entries for the week beginning with "date".

-m[*month***]** Print entries for the given "month".

By default, **calendar** print entries for today and tomorrow, with "tomorrow" encompassing the following Monday should "today" be a Friday or Saturday. If an entry in your **.calendar** has an at-sign '@' embedded in it, **calendar** prints it regardless of when it is to occur, until its date has passed.

The following gives an example of a calendar file. As you can see, **calendar** understands different formats of dates:

```
Apr 16     Dave's birthday
7/6        Dad's birthday
Sep 26     Mom's birthday
Jun 30     Barry's birthday
10/4       Marianne's birthday
Jul 31     Anniversary!
Mar 16     Pot luck luncheon
```

You can run **calendar** automatically by embedding the command

```
calendar
```

in your **.profile**.

If you wish, you can run **calendar** automatically for yourself, by inserting it into file **/usr/spool/cron/crontabs/root**. In this case, **calendar** should be used with its **-a** option, to force it to search each user's **$HOME** directory for **.calendar** and mail the appointments it finds to that user.

See Also

commands

Notes

calendar's notion of tomorrow understands weekends but not holidays. Thus, if you invoke **calendar** on a Friday, it returns the events for that day and the following Saturday, Sunday, and Monday. If Monday is a holiday, however, you will not receive appointments for Tuesday.

calling conventions — Definition

The following presents the calling conventions for COHERENT.

The calling conventions of C take into account machine architecture and the fact that the number of arguments passed to a function may vary, as in the functions **printf()** and **scanf()**.

For example, consider the following C program, called **foo.c**:

```
short a;
long b;
char c;

foo()
{
        example(a, b, c);
}
```

Compiling this program with the command

```
cc -S foo.c
```

generates the assembly-language code (with added comments):

```
  .alignoff
  .comm      a,          2       / a, b, and c are commons in the .bss
  .comm      b,          4
  .comm      c,          1
```

```
foo:
        .text
        .globl foo

foo:
        push      %ebp
        movl      %ebp, %esp
        movsxb    %eax, c             / move c to %eax with sign extend
        push      %eax                / pass c
        push      b                   / pass b
        movsxw    %eax, a             / move a to %eax with sign extend
        push      %eax                / pass a
        call      example

        leave                         / epilog code for foo
        ret
        .align    4
```

Note the following points:

- Parameters are pushed in reverse order. You should not depend on this feature, as the ANSI standard says that parameters may be calculated and pushed in any order.
- The stack is reset by the caller, not the callee. Only the caller knows the number of parameters pushed.
- All parameters become **int** or **double** when passed under Kernighan & Ritchie C. This changes under ANSI C.

Now consider the module **example.c**, which gives the receiving end:

```
double
example(x, y, z)
short x;
long y;
char z;
{
      int tmp;

      tmp = x * y;
      return (tmp + z);
}
```

The command

```
cc -S example.c
```

generates the code:

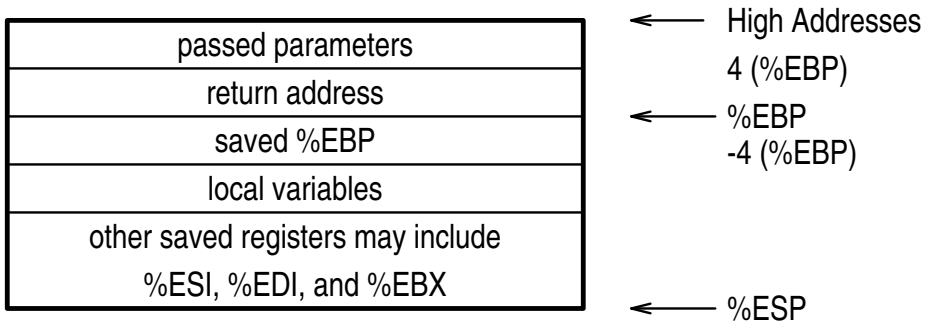
```
        .alignoff

        .text
        .globl example

example:
        enter     $4, $0              / 4 bytes of local variables
        push      %edi
        movl      %eax, 12(%ebp)      / x * y
        imull     8(%ebp)             / 8 == 4 + sizeof(int)
        movl      -4(%ebp), %eax      / save into tmp
        movl      %edi, 16(%ebp)      / tmp + z
        addl      %edi, %eax          / return double in EDX:EAX
        movl      %eax, %edi
        call      _dicvt
        pop       %edi

        leave                         / leave with result in %eax:%edx
        ret
        .align    4
```

After the prologue code, the stack always looks like



Notice that parameters start at

```
[4 + first parm size](%ebp)
```

and go to higher addresses, whereas local variables start at

```
-4(%ebp)
```

and go to lower addresses. Therefore, if you have a local array and overwrite it in the forward direction, you clobber your caller's **%ebp**; if you overwrite it in the backward direction, you clobber your caller's register variables (although if the caller has no register variable, it's harmless).

On the 80386, the stack starts at 0x80000000 and grows down being expanded by the system as it is needed. Reasonable programs should never have stack-overflow problems, as they did under COHERENT 286.

Note that the convention for returning floating-point numbers differ depending upon whether a program uses software floating-point emulation, or hardware floating-point code as invoked by the **cc** option **-VNDP**. Programs that use hardware floating point return **double** in the NDP stack top **$st0**.

See Also

C language, Programming COHERENT,

calloc() — General Function (libc)

Allocate dynamic memory
#include <stdlib.h>
char ***calloc(***count***,** *size***)**
unsigned *count***,** *size***;**

The function **calloc()** is one of a set of routines that helps manage a program's arena. **calloc()** calls **malloc()** to obtain a block large enough to contain *count* items of *size* bytes each; it then initializes the block to zeroes. When this memory is no longer needed, you can return it to the free pool by using the function **free()**.

calloc() returns the address of the chunk of memory it has allocated, or NULL if it could not allocate memory.

Example

This example attempts to **calloc()** a small portion of memory; it then reallocates it to demonstrate **realloc()**.

```
#include <stdio.h>
#include <stdlib.h>

main()
{
      register char *ptr, *ptr2;
      extern char *calloc(), *realloc();
      unsigned count, size;

      count = 4;
      size = sizeof(char *);
```

```
      if ((ptr = calloc(count, size)) != NULL)
            printf("%u blocks of size %u calloced\n",
                    count, size);
      else
            printf("Insuff. memory for %u blocks of size %u\n",
                 count, size);

      if ((ptr2 = realloc(ptr,(count*size) + 1)) != NULL)
            printf("1 block of size %u realloced\n",
                  (count*size)+1);
}
```

See Also

alloca(), arena, free(), libc, malloc(), memok(), realloc(), setbuf(), stdlib.h
ANSI Standard, §7.10.3.1
POSIX Standard, §8.1

Notes

The function **alloca()** allocates space on the stack. The space so allocated does not need to be freed when the function that allocated the space exits.

cancel — Command

Cancel a print job
cancel [*job* **[** *... job***]] [-all]**

The command **cancel** cancels execution of a printing job. It recognizes the following options:

-all Cancel all requests that are currently executing.

job Cancel each *job*. Each *job* is identified by the number printed by **lp** when the job was first spooled.

When a job is cancelled, it remains in the print queue for the remainder of its "lifetime", and may be printed later. When it cancels a job, **cancel** sends mail to the owner of the job to notify him of the job's cancellation.

cancel does not affect jobs that have already been downloaded into their destination printers. The only way to stop a job from printing after it has been downloaded is to clear the printer's memory. See the documentation that came with your printer for instructions on how to do that.

See Also

commands, lp, printer

Notes

cancel is a link to **lpstat**.

cancel is available only under COHERENT release 4.2 and subsequent releases.

canon.h — Header file

Portable layout of binary data
#include <canon.h>
#include <sys/types.h>

The routines declared in **canon.h** were designed to aid the transfer of binary information among different implementations of COHERENT. For technical reasons, these routines are slated to be dropped from a future release of COHERENT. Their use is strongly discouraged.

See Also

ar.h, byte ordering, header files,

captoinfo — Command

Convert termcap data to terminfo form
captoinfo [*filename***]**

The command **captoinfo** converts a file of terminal information that is in the **termcap** format into **terminfo** source format.

captoinfo reads *filename*; if no file is named on the command line, it reads the standard input. It writes its product to the standard output.

The input to **captoinfo** must be in correct **termcap** format. **captoinfo** complains about all constructs that it cannot interpret.

See Also

commands, termcap, terminfo, tic

Notes

The original code for **captoinfo** was written by Robert Viduya of the Georgia Institute of Technology, and was adapted for COHERENT by Mark Williams Company.

case — Command

Execute commands conditionally according to pattern
case *token* **in [***pattern* **[|** *pattern***] ...)** *sequence* **;;] ... esac**

case is a construct that used by the shell. It tells the shell to execute commands conditionally, according to a pattern. It tests the given *token* successively against each *pattern*, in the order given. It then 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.

Each *pattern* can include text 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.

Example

The following example prints a string in response to a command-line option:

```
case $1 in
      FOO)  echo "This is option FOO";;
      BAR)  echo "This is option BAR";;
      BAZ)  echo "This is option BAZ";;
      *)    echo "An asterisk marks the default option";;
esac
```

See Also

commands, ksh, sh

case — C Keyword

Introduce entry in switch statement

The C keyword **case** is a label within a **switch** statement. For example:

```
while ((int = getchar()) != EOF)
      switch (foo) {
      case 'q':
      case 'Q':
            exit(0);
      case ' ':
            n++;
      default:
            break;
      }
```

case labels each of the three possibilities recognized by the **switch** statement: a space, 'q', and 'Q'. The statements that follow a **case** statement behave as if they were enclosed within braces.

Note that a **case** statement is simply a label: it sets a point to which the **switch** statement jumps, and execution continues from that point. Once a **switch** statement jumps to the point marked by a given **case** label, execution continues until an **exit**, **break**, or **return** is read, or the closing brace of the **switch** statement is encountered.

See Also

break, C keywords, switch
ANSI Standard, §6.6.4.2

cast — Definition

The *cast* operation "coerces" a variable from one data type to another.

There are two reasons to cast a variable. The first is to convert a variable's data into a form acceptable to a given function. For example, the function **hypot** takes two **double**s. If the variables **leg_x** and **leg_y** are **float**s, the rules of C require that they be cast automatically to **double**. If the compiler did not do not do this, **hypot** would grab a **double**'s worth of memory: the four bytes of your **float**, plus four bytes of whatever happens to be sitting on the stack. The leads to results that are less than totally accurate.

The other reason to cast a variable is when you cast one type of pointer to another. For example,

```
char *foo;
int *bar;
bar = (int *)foo;
```

Although **foo** and **bar** are of the same length, you would cast **foo** in this instance to stop the C compiler from complaining about a type mismatch.

See Also

data formats, data types, Programming COHERENT

cat — Command

Concatenate the contents of a file to the standard output
cat [**-u**][*file ...*]

cat copies each *file* arguments to the standard output. A '-' tells **cat** to read 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).

See Also

commands

Notes

If you redirect **cat**'s the output to one of its input files, it will loop forever, reading from the file the text that it has just written into it: in effect, **cat** will chase its own tail endlessly.

caveat utilitor — Definition

Latin (sort of): "Let the user beware." Cf, "Heads up!" in the American dialect.

See Also

Using COHERENT

cc — Command

C compiler
cc [*compiler options***]** *file . . . .* **[***linker options***]**

cc is the program that compiles C programs. It guides files of source and object code through each phase of compilation and linking. **cc** has many options to assist in the compilation of C programs; in essence, however, all you need to do to produce an executable file from your C program is type **cc** followed by the name of the file or files that hold your program. **cc** checks whether the file names you give it are reasonable, selects the right phase for each file, and performs other tasks that ease the compilation of your programs.

How cc Works

cc works as follows:

- If a file ends in **.c**, **cc** assumes that it contains C code, and compiles it. The compiler generates a relocatable object module with the suffix **.o**.

- If the file has the suffix **.s**, **cc** assumes that it is a file of assembly language, and invokes the assembler **as** to assemble it. The assembler also generates a relocatable object module with the suffix **.o**.
- **cc** assumes that all files with the suffix **.o** are relocatable object modules. It also assumes that all files with the suffix **.a** are libraries of object modules. It passes both directly to the linker **ld**. Additional libraries can also be invoked by using the **-l** option **cc**, described below.
- Once all files of C code and assembly language have been compiled or assembled, **cc** then invokes the linker **ld** to link the newly created object files with any objects and libraries you named on **cc** command line. It also automatically includes the C runtime startup routine and the standard C library, so you do not have to name these on your **cc** command line.
- **cc** also cleans up after itself. It removes all of its temporary files automatically. If only one object file is created during compilation, **cc** deletes it after linking; however, if more than one object file is created, or if an object file of the same name existed before you began to compile, then the object file or files are not deleted.

Assuming that no error occurs along the way, **cc** writes the linked result into a file named after the *file* on its command line, minus that file's suffix — **.c**, **.s**, or **.o**, depending upon the type of data *file* holds. It is now ready to be executed.

Options

The following lists all of **cc**'s command-line options. **cc** passes some options through to the linker **ld** unchanged, and correctly interprets for it the options **-o** and **-u**.

A number of the options are esoteric and normally are not used when compiling a C program. The following are the most commonly used options:

-c	Compile only; do not link
-f	Link in floating-point **printf()**
-l*name*	Pass library **lib***name***.a** to linker
-o *name*	Call output file *name*
-V	Print verbose listing of **cc**'s action

? Print a detailed usage message that describes available **cc**'s options to the standard output.

-A MicroEMACS option. If an error occurs during compilation, **cc** automatically invokes the MicroEMACS screen editor. The error or errors are displayed in one window and the source code file in the other, with the cursor set to the line number indicated by the first error message. Typing **<ctrl-X>>** moves to the next error, **<ctrl-X><** moves to the previous error. To recompile, close the edited file with **<ctrl-Z>**. Compilation will continue either until the program compiles without error, or until you exit from the editor by typing **<ctrl-U>** followed by **<ctrl-X><ctrl-C>**.

-a By default, **cc** generates an executable file that is named after the source module. For example, the command

```
cc foo.c
```

generates an executable named **foo**. If you name more than source module on the **cc** command line, by default it names the executable after the first module you name. The option **-a** tells **cc** to create an executable file named **a.out**. This is for compatibility with other versions of UNIX. Note that option **-o**, described below, overrides the effect of **-a**.

-B[*path***]**

Backup option. Use an alternative *path* for the compiler phases **cc0**, **cc1**, **cc2**, and **cc3**. If *path* is supplied, **cc** prefixes it onto the name of each phase of the compiler, to form the name of the new compiler phase, and the path to the directory in which it lives. If you do not supply a *string*, **cc** prefixes the name of the current directory.

If you precede a **-B** option with a **-t** option, the **-B** option affects only the phase of the compiler that the **-t** option names. For example, the command

```
cc -t0 -B/usr/fred/bin hello.c
```

compiles **hello.c** using the version of **cc0** found in directory **/usr/fred/bin**. You can include any number of pairs of **-t** and **-B** options, with each **-t** option naming phase of the compiler that the subsequent **-B** option affects.

If followed by the prefix option **-M**, the name of the compiler phase in question is prefixed by the string named in the **-M** option. For example, the command

```
cc -t0 -B/usr/fred/cc -Mnew a.c
```

tells the compiler to look for **/usr/fred/cc/newcc0** and execute instead of the usual **cc0**.

-c Compile option. Suppress linking and the removal of the object files.

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

-E Expand option. Run the C preprocessor **cpp** and write its output onto the standard output.

-f Floating-point option. Include the version of **printf()** that converts floating-point numbers to text. If a program is compiled without the **-f** option but attempts to print a floating-point number during execution by using the **e**, **f**, or **g** format specifications to **printf()**, the program prints the error message

```
You must compile with -f option for floating point
```

and exits.

Note that if you wish to include the **libm** library routines that perform floating-point mathematics functions, you must specify **-lm** on the command line to load the library **libm.a**.

-g Generate debugging information. Same as option **-VDB**, described below.

-I*name*
Include option. Specify a directory the preprocessor should search for files given in **#include** directives, using the following criteria: If the **#include** statement reads

```
#include "file.h"
```

cc searches for **file.h** first in the source directory, then in the directory named in the **-I***name* option, and finally in the system's default directories. If the **#include** statement reads

```
#include <file.h>
```

cc searches for **file.h** first in the directories named in the **-I***name* option, and then in the system's default directories. Multiple **-I***name* options are executed in the order of their appearance.

-K Keep option. Do not erase the intermediate files generated during compilation. Temporary files will be written into the current directory.

-L*directory*
Tell the linker **ld** to search *directory* for its libraries before it searches the directories named in the environmental variable **LIBPATH**. You can use multiple **-L** options in a **cc** command.

-l*name*
Pass the name of a library to the linker. **cc** expands **-l***name* into **/lib/lib***name***.a**. If an alternative library prefix has been specified by the **-tl** and **-B***string* options, then **-l***name* expands to *string***lib***name***.a**. Note that this is a *linker option*, and so must appear at the end of the **cc** command line, or it will not be processed correctly.

-M*string*
Machine option. Use an alternate version of **cc0**, **cc1**, **cc1a**, **cc1b**, **cc2**, **cc3**, **as**, **lib*.a**, and **crts0.o,** named by fixing *string* between the directory name and the pass and file names. For examples, see the description of option **-B**, above. Before release 4.0 of COHERENT, **cc** executed the compiler phases **/lib/cc0** through **/lib/cc3**. Beginning with release 4.0, **cc** itself contains all the compiler phases; the preprocessor **/lib/cpp** executes the parser **/lib/cc0**, but compiler phases **/lib/cc[123]** do not exist for **cc**.

-o *name*
Output option. Rename the executable file from the default to *name*. Unlike UNIX, the COHERENT implementation of **cc** by default names an executable after the first first **.c** or **.o** file given on the command line, instead of naming it **a.out**. If you want **cc** to conform to the UNIX standard, set include the option **-o a.out** when you set the environmental variable **CCHEAD**. This environmental variable is described below. Another approach is to invoke **make** to control compilation. For details, see the Lexicon entry for **make**.

-O Optimize option. Run the code generated by the C compiler through the peephole optimizer. The optimizer pass is mandatory for the i8086, Z8000, and M68000 compilers, and need not be requested. It is optional for the PDP-11 compiler, but is recommended for all files except those that consist entirely of initialized tables of data.

-p Generate code to profile functions calls. Programs compiled with this option can be run with the command **prof** to print a summary of how much time the program spends in each subroutine, to help you optimize your programs. You must use this option to compile each module whose functions you wish to examine; and you must also use this option on the **cc** command line with which you link the program, to ensure that the appropriate library routines are linked into your executable.

-Q Quiet option. Suppress all messages, no matter how awful an error they indicate.

-S Suppress the object-writing and link phases, and invoke the disassembler **cc3**. This option produces an assembly-language version of a C program for examination, for example if a compiler problem is suspected. The assembly-language output file name replaces the **.c** suffix with **.s**. This is equivalent to the **-VASM** option.

-T*size*

cc writes its temporary data into two 64-kilobytes buffers that grow as needed. The **-T** option tells **cc** to use buffers of *size* bytes each. Setting these to a larger size may help large files compile faster. Setting *size* to zero forces **cc** to use temporary files written onto the disk.

-t*phase*

Take option. Use an alternate versions of the phase or phases of the compiler specified by **phase**, which must consist of one or more of the characters **01ab23sdlrt**. If no *phase* string appears, **cc** uses alternate version of every phase of the compiler, except the preprocessor. If the **-t** option is followed by a **-B** option, **cc** prefixes the path named in the **-B** option to the phases and files named in the **-t** option. For examples, see the description of option **-B**, above,

-U*name*

Undefine symbol *name*. Use this option to undefine symbols that the preprocessor defines implicitly, such as the name of the native system or machine. Users who wants serious ISO namespace compliance should compile with the options:

```
-UCOHERENT -UMWC -U_I386 -U_IEEE
```

These options turn off the macros **COHERENT**, **MWC**, **_I386**, and **_IEEE**, all of which are automatically defined by the COHERENT preprocessor.

-V Verbose option. **cc** prints onto the standard output a step-by-step description of each action it takes.

-V*string*

Variant option. Toggle (i.e., turn on or off) the variant *string* during the compilation. Variants that are marked **on** are turned on by default. Options marked **Strict:** generate messages that warn of the conditions in question. **cc** recognizes the following variants:

-VASM
Output assembly-language code. Identical to **-S** option, above. Default is **off**.

-VCOMM
Permit **.com**-style data items. Default is **on**.

-VCPLUS
Ignore C++-style comments, which are deliminted by '//'.

-VDB
Generate debugging information, same as option **-g** described above. Default is **off**.

-VFLOAT
Include floating-point **printf()** code. Same as option **-f**, described above.

-VNDP
Generate code to execute hardware floating-point arithmetic. **cc** executes floating-point arithmetic on an 80387 or 80486-DX, if present; or use software emulation if it is not. For more information, see the section on hardware floating-point arithmetic, below.

-VNOWARN
Suppress all warning and strict messages. Use this option if you wish to suppress cascades of warning message about, say, nested comments.

-VPROF
Same as the option **-p**, described above.

-VPSTR
"imPure" strings: Place all string literals into the **.data** segment rather than in **.text**. This may be necessary for sloppily written code that assumes it can overwrite string literals.

-VQUIET
Suppress all messages. Identical to **-Q** option. Default is **off**.

-VS Turn on all strict checking. Default is **on**.

-VSBOOK
Strict: note deviations from *The C Programming Language*, ed. 1. Default is **off**.

-VSCCON
Strict: note constant conditional. Default is **off**.

-VSINU
Implement struct-in-union rules instead of Berkeley-member resolution rules. Default is **off**, i.e., Berkeley rules are the default.

-VSLCON
Strict: **int** constant promoted to **long** because value is too big. Default is **on**.

-VSMEMB
Strict: check use of structure/union members for adherence to standard rules of C. Default is **on**.

-VSNREG
Strict: register declaration reduced to auto. Default is **on**.

-VSPVAL
Strict: pointer value truncated. Default is **off**.

-VSRTVC
Strict: risky types in truth contexts. Default is **off**.

-VSTAT
Give statistics on optimization.

-VSUREG
Strict: note unused registers. Default is **off**.

-VSUVAR
Strict: note unused variables. Default is **on**.

-VVERSION
Print to the standard error the compiler's version number. This information is useful when reporting bugs.

-VWIDEN
Warn the user if a parameter is widened from **char** or **short** to **int**, or from **float** to **double**. Default is **off**.

-V3GRAPH
Translate ANSI trigraphs. Default is **off**.

cc reads the environmental variables **CCHEAD** and **CCTAIL** and appends their contents to, respectively, the beginning and the end of the **cc** command. For example, if you insert the following entries into your **.profile**

```
export CCHEAD='-f -o a.out'
export CCTAIL=-lm
```

then **cc** will always use the floating-point version of **printf()**, always write its executable into file **a.out**, and always link in the mathematics library **libm**. In effect, it turns the command

```
cc hello.c
```

into:

```
cc -f -o a.out hello.c -lm
```

If you set a command option in **CCHEAD** or **CCTAIL**, you can always override it for specific **cc** commands. For example, if you have set **-o a.out** in **CCHEAD**, typing the command

```
cc -o hello hello.c
```

generates the command:

```
cc -o a.out -o hello hello.c
```

The latter **-o** option is the one used, and in effect cancels the effect of the **CCHEAD** entry. Thus, setting **CCHEAD** and **CCTAIL** give you a flexible way to set **cc**'s default behavior.

Note that

```
CCHEAD='-Wa,-f -Wl,-oa.out'
```

will give you a compilation environment that matches that of the UNIX operating system.

Linking Objects

The linker **ld** does not know about paths: it links exactly what you tell it to link via the **cc** command line. **cc** looks for compiler phases and for runtime startoff and library by searching the directories named in the environmental variable **LIBPATH**. If you do not define **LIBPATH** in your environment, it searches the default **LIBPATH** as defined in **/usr/include/path.h**. If you define **LIBPATH**, **cc** searches the directories in the order you specify. For example, a typical definition is:

```
export LIBPATH=:/lib:/usr/lib
```

This searches the current directory '.', then **/lib**, then **/usr/lib**.

Hardware Floating-Point Arithmetic

The C compiler shipped with version of COHERENT prior to release 4.2 generated software floating-point calls. That is, floating-point code such as

```
d1 = d2 + 2.5;
```

generated calls to software routines to perform the desired operations. This is called "software floating-point arithmetic".

Beginning with release 4.2.05 of COHERENT, **cc** generates software floating-point arithmetic by default, but let you select "hardware floating-point arithmetic". With hardware floating-point arithmetic, **cc** generates calls to execute floating-point operations on a numeric data processor (NDP), such as the 80387. To do so, use the option **-VNDP**. A program compiled to perform hardware floating-point arithmetic runs correctly on any computer: if the computer contains an NDP, the code executes on that part; but if the computer does not contain an NDP, the code emulates the operation of the NDP. Note that persons who do *not* have an NDP on their system must have the floating-point emulation module linked into their kernels; those who do have an NDP, however, do not need this module. The libraries in directories **/lib** and **/usr/lib** are compiled using software floating-point arithmetic; the libraries compiled with hardware floating-point arithmetic are kept in sub-directories **/lib/ndp** and **/usr/lib/ndp**.

As mentioned above, code compiled to use hardware floating-point arithmetic runs much faster when your machine has an NDP installed. If your system does not have a numeric co-processor (i.e., an 80387, 80487, an 80486DX, or a Pentium) and you wish to run programs that intensively use floating-point arithmetic, we strongly urge you to consider upgrading your system to use an NDP.

Files

/bin/cc — C compiler

See Also

as, C language, cc0, cc1, cc2, cc3, commands, C preprocessor, cpp, ld, LIBPATH, make, makedepend, TMPDIR
The C Language tutorial

Diagnostics

The following gives the error messages returned by the COHERENT C compiler. The messages are in alphabetical order, and each is marked as to whether it is a *fatal*, *error*, *warning*, or *strict* condition. A fatal message usually indicates a condition that caused the compiler to terminate execution. Fatal errors from the later phases of compilation often cannot be fixed, and may indicate problems in the compiler or assembler. An error message points to a condition in the source code that the compiler cannot resolve. This almost always occurs when the program does something illegal, e.g., has unbalanced braces. Warning messages point out code that is compilable, but may produce trouble when the program is executed. A strict message refers to a passage in the code that is unorthodox and may not be portable. For error messages produced by the assembler **as**, the linker **ld**, and the preprocessor **cpp**, see their respective entries in the Lexicon.

ambiguous reference to "*string*" *(error)*
: *string* is defined as a member of more than one **struct** or **union**, is referenced via a pointer to one of those **struct**s or **union**s, and there is more than one offset that could be assigned.

argument list has incorrect syntax *(error)*
: The argument list of a function declaration contains something other than a comma-separated list of formal parameters.

array bound must be a constant *(error)*
: An array's size can be declared only with a constant; you cannot declare an array's size by using a variable. For example, it is correct to say **foo[5]**, but illegal to say

```
bar = 5;
foo[bar];
```

array bound must be positive *(error)*
: An array must be declared to have a positive number of elements. The array flagged here was declared to have a negative size, e.g., **foo[-5]**.

array bound too large *(error)*
: The array is too large to be compiled with 32-bit index arithmetic. You should devise a way to divide the array into compilable portions.

array row has 0 length *(error)*
: This message can be triggered by either of two problems. The first problem is declaring an array to have a length of zero; e.g., **foo[0]**. The second problem is failing to declare the size of a dimension *other than the first* in a multi-dimensional array. C allows you to declare an indefinite number of array elements of *n* bytes each, but you cannot declare *n* array elements of an indefinite length. For example, it is correct say **foo[][5]** but illegal to say **foo[5][]**.

associative expression too complex *(fatal)*
: An expression that uses associative binary operators (e.g., '+') has too many operators; for example, **i=i1+i2+i3+ . . . +i30;**. You should simplify the expression.

bad argument storage class *(error)*
: An argument was assigned a storage class that the compiler does not recognize. The only valid storage class is **register**.

bad external storage class *(error)*
: An **extern** has been declared with an invalid storage class, e.g., **register** or **auto**.

bad field width *(error)*
: A field width was declared either to be negative or to be larger than the object that holds it. For example, **char foo:9** or **char foo:-1** will trigger this error.

bad filler field width *(error)*
: A filler field width was declared either to be negative or to be larger than the object that holds it. For example, **char foo:9** or **char foo:-1** will trigger this error.

bad flexible array declaration *(error)*
: A flexible array is missing an array boundary; e.g., **foo[5][]**. C permits you to declare an indefinite number of array elements of *n* bytes each, but you cannot declare an array to have *n* elements of an indefinite number of bytes each.

break not in a loop *(error)*
A **break** occurs that is not inside a loop or a **switch** statement.

call of non function *(error)*
What the program attempted to call is not a function. Check to make sure that you have not accidentally declared a function as a variable; e.g., typing **char *foo;** when you meant **char *foo();**.

cannot add pointers *(error)*
The program attempted to add two pointers. **int**s or **long**s may be added to or subtracted from pointers, and two pointers to the same type may be subtracted, but no other arithmetic operations are legal on pointers.

cannot apply unary '&' to a register variable *(error)*
Because register variables are stored within registers, they do not have addresses, which means that the unary **&** operator cannot be used with them.

cannot cast double to pointer *(error)*
The program attempted to cast a **double** to a pointer. This is illegal.

cannot cast pointer to double *(error)*
The program attempted to cast a pointer to a **double**. This is illegal.

cannot cast structure or union *(error)*
The program attempted to cast a **struct** or a **union**. This is illegal.

cannot cast to structure or union *(error)*
The program attempted to cast a variable to a **union** or **struct**. This is illegal.

cannot declare array of functions *(error)*
For example, the declaration **extern int (*f)[]();** declares **f** to be an array of pointers to functions that return **int**s. Arrays of functions are illegal.

cannot declare flexible automatic array *(error)*
The program does not explicitly declare the number of elements in an automatic array.

cannot initialize fields *(error)*
The program attempted to initialize bit fields within a structure. This is not supported.

cannot initialize unions *(error)*
The program attempted to initialize a **union** within its declaration. **union**s cannot be initialized in this way.

string: cannot reopen *(fatal)*
The optimizer cannot reopen a file with which it has worked. Make sure that your mass storage device is working correctly and that it is not full.

case not in a switch *(error)*
The program uses a **case** label outside of a **switch** statement. See the Lexicon entry for **case**.

character constant overflows long *(error)*
The character constant is too large to fit into a **long**. It should be redefined.

character constant promoted to long *(warning)*
A character constant has been promoted to a **long**.

class not allowed in structure body *(error)*
A storage class such as **register** or **auto** was specified within a structure.

compound statement required *(error)*
A construction that requires a compound statement does not have one, e.g., a function definition, array initialization, or **switch** statement.

constant expression required *(error)*
The expression used with a **#if** statement cannot be evaluated to a numeric constant. It probably uses a variable in a statement rather than a constant.

constant "*number*" promoted to long *(warning)*
The compiler promoted a constant in your program to **long**; although this is not strictly illegal, it may create problems when you attempt to port your code to another system, especially if the constant appears

in an argument list.

constant used in truth context *(strict)*
A conditional expression for an **if**, **while**, or **for** statement has turned out to be always true or always false. For example, **while(1)** will trigger this message.

construction not in Kernighan and Ritchie *(strict)*
This construction is not found in *The C Programming Language*; although it can be compiled by COHERENT, it may not be portable to another compiler.

continue not in a loop *(error)*
The program uses a **continue** statement that is not inside a **for** for **while** loop.

declarator syntax *(error)*
The program used incorrect syntax in a declaration.

default label not in a switch *(error)*
The program used a **default** label outside a **switch** construct. See the Lexicon entry for **default**.

divide by zero *(warning)*
The program will divide by zero if this code is executed. Although the program can be parsed, this statement may create trouble if executed.

duplicated case constant *(error)*
A **case** value can appear only once in a **switch** statement. See the Lexicon entries for **case** and **switch**.

empty switch *(warning)*
A **switch** statement has no **case** labels and no **default** labels. See the Lexicon entry for **switch**.

error in enumeration list syntax *(error)*
The syntax of an enumeration declaration contains an error.

error in expression syntax *(error)*
The parser expected to see a valid expression, but did not find one.

exponent overflow in floating point constant *(warning)*
The exponent in a floating point constant has overflowed. The compiler has set the constant to the maximum allowable value, with the expected sign.

exponent underflow in floating point constant *(warning)*
The exponent in a floating point constant has underflowed. The compiler has set the constant to zero, with the expected sign.

expression too complex *(fatal)*
The code generator cannot generate code for an expression. You should simplify your code.

external syntax *(error)*
This could be one of several errors, most often a missing '{'.

file ends within a comment *(error)*
The source file ended in the middle of a comment. If the program uses nested comments, it may have mismatched numbers of begin-comment and end-comment markers. If not, the program began a comment and did not end it, perhaps inadvertently when dividing by ******something*, e.g., **a=b/*cd;**.

function cannot return a function *(error)*
The function is declared to return another function, which is illegal. A function, however, can return a *pointer* to a function, e.g., **int (*signal(n, a))()**

function cannot return an array *(error)*
A function is declared to return an array, which is illegal. A function, however, can return a pointer to a structure or array.

functions cannot be parameters *(error)*
The program uses a function as a parameter, e.g., **int q(); x(q);**. This is illegal.

identifier "*string*" is being redeclared *(error)*
The program declares variable *string* to be of two different types. This often is due to an implicit declaration, which occurs when a function is used before it is explicitly declared. Check for name conflicts.

LEXICON

identifier "*string*" is not a label *(error)*
The program attempts to **goto** a nonexistent label.

identifier "*string*" is not a parameter *(error)*
The variable "*string*" did not appear in the parameter list.

identifier "*string*" is not defined *(error)*
The program uses identifier *string* but does not define it.

identifier "*string*" not usable *(error)*
string is probably a member of a structure or **union** which appears by itself in an expression.

illegal character constant *(error)*
A legal character constant consists of a a backslash '\' followed by **a**, **b**, **f**, **n**, **r**, **t**, **v**, **x**, or up to three octal digits.

illegal character (*number* decimal) *(error)*
A control character was embedded within the source code. *number* is the decimal value of the character.

illegal # construct *(error)*
The parser recognizes control lines of the form **#***line_number* (decimal) or **#***file_name*. Anything else is illegal.

illegal integer constant suffix *(error)*
Integer constants may be suffixed with **u**, **U**, **l**, or **L** to indicate **unsigned**, **long**, or **unsigned long**.

illegal label "*string*" *(error)*
The program uses the keyword *string* as a **goto** label. Remember that each label must end with a colon.

illegal operation on "void" type *(error)*
The program tried to manipulate a value returned by a function that had been declared to be of type **void**.

illegal structure assignment *(error)*
The structures have different sizes.

illegal subtraction of pointers *(error)*
A pointer can be subtracted from another pointer only if both point to objects of the same size.

illegal use of a pointer *(error)*
A pointer was used illegally, e.g., multiplied, divided, or &-ed. You may get the result you want if you cast the pointer to a **long**.

illegal use of a structure or union *(error)*
You may take the address of a **struct**, access one of its members, assign it to another structure, pass it as an argument, and return. All else is illegal.

illegal use of floating point *(error)*
A **float** was used illegally, e.g., in a bit-field structure.

illegal use of "void" type *(error)*
The program used **void** improperly. Strictly, there are only **void** functions; COHERENT also supports the cast to **void** of a function call.

illegal use of void type in cast *(error)*
The program uses a pointer where it should be using a variable.

inappropriate signed *(error)*
The **signed** modifier may only be applied to **char**, **short**, **int**, or **long** types.

inappropriate "long" *(error)*
Your program used the type **long** inappropriately.

inappropriate "short" *(error)*
Your program used the type **short** inappropriately.

inappropriate "unsigned" *(error)*
Your program used the type **unsigned** inappropriately.

indirection through non pointer *(error)*
The program attempted to use a scalar (e.g., a **long** or **int**) as a pointer. This may be due to not dereferencing the scalar.

initializer too complex *(error)*
An initializer was too complex to be calculated at compile time. You should simplify the initializer to correct this problem.

integer pointer comparison *(strict)*
The program compares an integer or **long** with a pointer without casting one to the type of the other. Although this is legal, the comparison may not work on machines with non-integer size pointers, e.g., Z8001 or LARGE-model on the i8086 family, or on machines with pointers larger than **int**s, e.g., the M68000 family of microprocessors.

integer pointer pun *(strict)*
The program assigns a pointer to an integer, or vice versa, without casting the right-hand side of the assignment to the type of the left-hand side. For example,

```
char *foo;
long bar;
foo = bar;
```

Although this is permitted, it is often an error if the integer has less precision than the pointer does. Make sure that you properly declare all functions that returns pointers.

internal compiler error *(fatal)*
The program produced a state that should not happen during compilation. Try to localize the offending statement if at all possible. Forward a minimal program that exhibits the error, preferably on a machine-readable medium, to Mark Williams Company, together with the version number of the compiler, the command line used to compile the program, and the system configuration. For immediate advice during business hours, telephone Mark Williams Company technical support.

"*string*" is a enum tag *(error)*
"*string*" is a struct tag *(error)*
"*string*" is a union tag *(error)*
string has been previously declared as a tag name for a **struct**, **union**, or **enum**, and is now being declared as another tag. Perhaps the structure declarations have been included twice.

"*string*" is not a tag *(error)*
A **struct** or **union** with tag *string* is referenced before any such **struct** or **union** is declared. Check your declarations against the reference.

"*string*" is not a typedef name *(error)*
string was found in a declaration in the position in which the base type of the declaration should have appeared. *string* is not one of the predefined types or a **typedef** name. See the Lexicon entry on **typedef** for more information.

"*string*" is not an "enum" tag *(error)*
An **enum** with tag *string* is referenced before any such **enum** has been declared. See the Lexicon entry for **enum** for more information.

class "*string*" [*number*] is not used *(strict)*
Your program declares variable *string* or *number* but does not use it.

label "*string*" undefined *(error)*
The program does not declare the label *string*, but it is referenced in a **goto** statement.

left side of "*string*" not usable *(error)*
The left side of the expression *string* should be a pointer, but is not.

lvalue required *(error)*
The left-hand value of a declaration is missing or incorrect. See the Lexicon entries for **lvalue** and **rvalue**.

member "*string*" is not addressable *(error)*
The array *string* has exceeded the machine's addressing capability. Structure members are addressed with 16-bit signed offsets on most machines.

member "*string*" is not defined *(error)*
The program references a structure member that has not been declared.

mismatched conditional *(error)*
In a '?:' expression, the colon and all three expressions must be present.

misplaced ":" operator *(error)*
The program used a colon without a preceding question mark. It may be a misplaced label.

missing "(" *(error)*
The **if**, **while**, **for**, and **switch** keywords must be followed by parenthesized expressions.

missing "=" *(warning)*
An equal sign is missing from the initialization of a variable declaration. Note that this is a warning, not an error: this allows COHERENT to compile programs with "old style" initializers, such as **int i 1**. Use of this feature is strongly discouraged, and it will disappear when the ANSI standard for the C language is adopted in full.

missing "," *(error)*
A comma is missing from an enumeration member list.

missing ":" *(error)*
A colon ':' is missing after a **case** label, after a default label, or after the '?' in a '?'-':' construction.

missing ";" *(error)*
A semicolon ';' does not appear after an external data definition or declaration, after a **struct** or **union** member declaration, after an automatic data declaration or definition, after a statement, or in a **for(;;)** statement.

missing "]" *(error)*
A right bracket ']' is missing from an array declaration, or from an array reference; for example, **foo[5**.

missing "{" *(error)*
A left brace '{' is missing after a **struct** *tag*, **union** *tag*, or **enum** *tag* in a definition.

missing "}" *(error)*
A right brace '}' is missing from a **struct**, **union**, or **enum** definition, from an initialization, or from a compound statement.

missing "while" *(error)*
A **while** command does not appear after a **do** in a **do**-**while()** statement.

missing label name in goto *(error)*
A **goto** statement does not have a label.

missing member *(error)*
A '.' or '->' is not followed by a member name.

missing right brace *(error)*
A right brace is missing at end of file. The missing brace probably precedes lines with errors reported earlier.

missing "*string*" *(error)*
The parser **cc0** expects to see token *string*, but sees something else.

missing semicolon *(error)*
External declarations should continue with ',' or end with ';'.

missing type in structure body *(error)*
A structure member declaration has no type.

multiple classes *(error)*
An element has been asigned to more than one storage class, e.g., **extern register**.

multiple types *(error)*
An element has been assigned more than one data type, e.g., **int float**.

nonterminated string or character constant *(error)*
A line that contains single or double quotation marks left off the closing quotation mark. A newline in a string constant may be escaped with '\'.

number has too many digits *(error)*
A number is too big to fit into its type.

only one default label allowed *(error)*
The program uses more than one **default** label in a **switch** expression. See the Lexicon entries for **default** and **switch** for more information.

out of tree space *(fatal)*
The compiler allows a program to use up to 350 tree nodes; the program exceeded that allowance.

parameter *string* is not addressable *(error)*
The parameter has a stack frame offset greater than 32,767. Perhaps you should pass a pointer instead of a structure.

potentially nonportable structure access *(strict)*
A program that uses this construction may not be portable to another compiler.

return type/function type mismatch *(error)*
What the function was declared to return and what it actually returns do not match, and cannot be made to match.

return(e) illegal in void function *(error)*
A function that was declared to be type **void** has nevertheless attempted to return a value. Either the declaration or the function should be altered.

risky type in truth context *(strict)*
The program uses a variable declared to be a pointer, **long**, **unsigned long**, **float**, or **double** as the condition expression in an **if**, **while**, **do**, or '?'-':'. This could be misinterpreted by some C compilers.

size of *string* overflows size_t *(strict)*
A string was so large that it overran an internal compiler limit. You should try to break the string in question into several small strings.

size of union "*string*" is not known *(error)*
A pointer to a **struct** or **union** is being incremented, decremented, or subjected to array arithmetic, but the **struct** or **union** has not been defined.

size of *string* too large *(error)*
The program declared an array or **struct** that is too big to be addressable, e.g., **long a[20000];** on a machine that has a 64-kilobyte limit on data size and four-byte **long**s.

sizeof truncated to unsigned *(warning)*
An object's **sizeof** value has lost precision when truncated to a **size_t** integer.

sizeof(*string*) set to *number (warning)*
The program attempts to set the value of *string* by applying **sizeof** to a function or an **extern**; the compiler in this instance has set *string* to *number*.

storage class not allowed in cast *(error)*
The program **cast**s an item as a **register**, **static**, or other storage class.

string initializer not terminated by NUL *(warning)*
An array of **char**s that was initialized by a string is too small in dimension to hold the terminating NUL character. For example, **char foo[3] = "ABC"**.

structure "*string*" does not contain member "*m*" *(error)*
The program attempted to address the variable *string.m*, which is not defined as part of the structure *string*.

structure or union used in truth context *(error)*
The program uses a structure in an **if**, **while**, or **for**, or '?:' statement.

switch of non integer *(error)*
The expression in a **switch** statement is not type **int** or **char**. You should cast the **switch** expression to an **int** if the loss of precision is not critical.

switch overflow *(fatal)*
The program has more than ten nested **switch**es.

too many adjectives *(error)*
A variable's type was described with too many of **long**, **short**, or **unsigned**.

too many arguments *(fatal)*
No function may have more than 30 arguments.

too many cases *(fatal)*
The program cannot allocate space to build a **switch** statement.

too many initializers *(error)*
The program has more initializers than the space allocated can hold.

too many structure initializers *(error)*
The program contains a structure initialization that has more values than members.

trailing "," in initialization list *(warning)*
An initialization statement ends with a comma, which is legal.

type clash *(error)*
The parser expected to find matching types but did not. For example, the types of **e1** and **e2** in **(x) ? e1 : e2** must either both be pointers or neither be pointers.

type of function "*string*" adjusted to *string (warning)*
This warning is given when the type of a numeric constant is widened to **unsigned**, **long**, or **unsigned long** to preserve the constant's value. The type of the constant may be explicitly specified with the **u** or **L** constant suffixes.

type of parameter "*string*" adjusted to *string (warning)*
The program uses a parameter that the C language says must be adjusted to a wider type, e.g., **char** to **int** or **float** to **double**.

type required in cast *(error)*
The type is missing from a cast declaration.

unexpected end of enumeration list *(error)*
An end-of-file flag or a right brace occurred in the middle of the list of enumerators.

unexpected EOF *(fatal)*
EOF occurred in the middle of a statement. The temporary file may have been corrupted or truncated accidentally. Check your disk drive to see that it is working correctly.

union "*string*" does not contain member *m (error)*
The program attempted to address the variable string *m*, which is not defined as part of the structure *string*.

write error on output object file *(fatal)*
cc could not write the relocatable object module. Most likely, your mass storage device has run out of room. Check to see that your disk drive or hard disk has enough room to hold the object module, and that it is working correctly.

zero modulus *(warning)*
The program will perform a modulo operation by zero if the code just parsed is executed. Although the program can be parsed, this statement may create trouble if executed.

Notes

If you see the message

```
Out of memory
```

when compiling, this probably means that your program has exhausted the buffer space available to it. Use the option **-T0** to force **cc** to write its temporary files on the disk.

Prior to COHERENT release 4.2, **cc** wrote its diagnostic messages to the standard output device. **cc** now writes its diagnostic messages to the standard error. You may need to modify any scripts that redirect the output of **cc**.

cc0 — Definition

cc0 is the parser for the COHERENT C compiler **cc**. It parses C programs using the method of recursive descent and translates the program into a logical tree format.

See Also

cc, cc1, cc2, cc3, cpp, Programming COHERENT

cc1 — Definition

cc1 is the code generator for the COHERENT C compiler. This phase generates code from the trees created by the parser, **cc0**. The code generation is table driven, with entries for each operator and addressing mode.

See Also

cc, cc0, cc2, cc3, cpp, Programming COHERENT

cc2 — Definition

cc2 is the optimizer/object generator phase of the COHERENT C compiler. It optimizes the code generated by **cc1**, and writes the object code. COHERENT uses multiple optimization algorithms. One optimizes jump sequences: it eliminates common code, optimizes span-dependent jumps, and removes jumps to jumps. The other function scans the generated code repeatedly to eliminate unnecessary instructions.

See Also

cc, cc0, cc1, cc3, cpp, Programming COHERENT

cc3 — Definition

cc3 is the output phase of the COHERENT C compiler. It writes a file of assembly language rather than a relocatable object module. This phase is optional; it allows you to examine the code generated by the compiler. To produce an assembly-language output of a C program, use the **-S** option on the **cc** command line. For example,

```
cc -S foo.c
```

tells **cc** to produce a file of assembly language called **foo.s**, instead of an object module.

See Also

cc, cc0, cc1, cc2, cpp, Programming COHERENT

CCHEAD — Environmental Variable

Append options to beginning of cc command line
export CCHEAD=*options*

The COHERENT compiler **cc** reads the environmental variables **CCHEAD** and **CCTAIL** before it begins its work. You can set these variables to hold the default options that you want the compiler always to use.

cc appends the options in **CCHEAD** to the beginning of its command line.

See Also

cc, CCTAIL, environmental variables

CCTAIL — Environmental Variable

Append options to end of cc command line
export CCTAIL=*options*

The COHERENT compiler **cc** reads the environmental variables **CCHEAD** and **CCTAIL** before it begins its work. You can set these variables to hold the default options that you want the compiler always to use.

cc appends the options in **CCTAIL** to the end of its command line.

See Also

cc, CCHEAD, environmental variables

cd — Command

Change directory
cd *directory*

The shell keeps track of the directory in which the user is currently working. If a command is not specified by a complete path name beginning with '/', the shell prefixes it with the name of the current working directory. **cd** changes the current working directory to *directory*. If no *directory* is specified, the directory named in the **$HOME** environmental variable becomes the current working directory.

See Also

commands, ksh, pwd, sh

CD-ROM — Overview

COHERENT support for read-only compact disk devices

The term *CD-ROM* stands for "compact disk — read-only memory". COHERENT supports a variety of CD-ROM devices, from which you can read files or play music.

Devices Supported

As of this writing, COHERENT supports three varieties of CD-ROM drives:

- Sony CD-ROM models CDU31A or CDU33A, plugged its own dedicated controller.
- Mitsumi CD-ROM models FX001, FX001 high speed, FX001D, or LU005, plugged into its own dedicated controller. Mitsumi model FX001 also is known to work when plugged into the CD-ROM port of the SoundblasterPro sound card; the other Mitsumi drives have not yet been tested with the Soundblaster Pro card.
- Any SCSI CD-ROM drive plugged into an Adaptec 1542 SCSI controller.
- Any SCSI CD-ROM drive plugged Seagate host adapter models ST01 or ST02.

Please note that the NEC SCSI CD-ROM is support for ISO file systems, but *not* for audio disks. That is because the NEC drive does not use a standard interface for audio disks.

To use the driver for the Sony CDU31A drive, you must build a kernel that contains the driver **cdu31**. Normally, this is done when you install or update COHERENT. To add the driver to the kernel after installation or updating, do the following:

- Log in as the superuser **root**.
- **cd** to directory **/etc/conf**.
- Execute script **cdu31/mkdev**. This script will walk you through the process of adding the driver to the kernel. If you are unsure of the answer to any question that the script asks you, select the default; in most instances, this is correct.
- Execute the command:

  ```
  /etc/conf/bin/idmkcoh -o coh.test
  ```

  This builds a new kernel called **coh.test**.
- Boot the new kernel, as described in the Lexicon entry **booting**.

To use the driver for the Mitsumi drive, you must build a kernel that contains the driver **mcd**. Normally, this is done when you install or update COHERENT. To add the driver to the kernel after installation or updating, do the following:

- Log in as the superuser **root**.
- **cd** to directory **/etc/conf**.
- Execute script **mcd/mkdev**. This script will walk you through the process of adding the driver to the kernel. If you are unsure of the answer to any question that the script asks you, select the default; in most instances, this is correct.

- Execute the command:

```
/etc/conf/bin/idmkcoh -o coh.test
```

  This builds a new kernel called **coh.test**.

- Boot the new kernel, as described in the Lexicon entry **booting**.

If your CD-ROM drive is attached to an Adaptec 1542 SCSI controller, you must modify the driver **hai** to support the drive. Do so as follows:

- Log in as the superuser **root**.
- **cd** to directory **/etc/conf**.
- Execute script **hai/mkdev**. This script will walk you through the process of configuring **hai** to support your SCSI devices. If you are already using **hai** to support a SCSI disk or SCSI tape, be sure that you do not alter how they are configured. If you are unsure of the answer to any question that the script asks you, select the default; in most instances, this is correct.
- Execute the command:

```
/etc/conf/bin/idmkcoh -o coh.test
```

  This builds a new kernel called **coh.test**.

- Boot the new kernel, as described in the Lexicon entry **booting**.

Reading a CD-ROM

COHERENT at present includes three commands for manipulating CD-ROMs: **cdview**, **cdv**, and **cdplayer**.

cdplayer lets you play audio CDs on your CD-ROM drive. It uses a text-based interface to let you display the contents of a CD, select a track, set the volume, and otherwise manipulate your audio CDs.

cdv is a script with which you can play CD-ROM disks — that is, disks that hold an ISO-9660 file system. The interface is character-based and rather crude; however, with it you can read the contents of a directory on a CD-ROM, or copy a file from the CD-ROM into a COHERENT directory. **cdview** is a lower-level command that is invoked through **cdv**.

Files

/dev/cdrom — Device applications read by default for CD-ROMs
/dev/rscd0 — Device for accessing Sony CDU31A CD-ROM
/dev/rmcd0 — Device for accessing Mitsumi CD-ROM
/dev/Scdrom* — Block-special SCSI CD-ROM devices
/dev/rScdrom* — Character-special SCSI CD-ROM devices

See Also

Administering COHERENT, cdplayer, cdrom.h, cdv, cdview, device drivers, hai, mcd

Notes

At present, you cannot mount an ISO-9660 file system onto your COHERENT system. A future release of COHERENT will permit you to do so.

Please note that COHERENT, like most UUCP-like operating systems, does not support playing audio CDs on a NEC/Toshiba CD-ROM. This is because NEC uses a non-standard interface for audio CDs.

cdmp — Command

Dump COFF files into a readable form
cdmp [-adlrs] *filename*

cdmp dumps a file in COFF format into its most readable format. Its default is to dump all information; but as this can produce a very large output file, **cdmp** lets you use the following switches to mix-and-match its output:

-a Suppress auxiliary symbol entries.

-d Suppress data dumps

-l Suppress line numbers.

-r Suppress relocation entries.

-s Suppress symbol entries.

cc and **as** do not produce line numbers and auxiliary-symbol entries, and **ld** does not preserve them.

cdmp writes its dump into the "vertical hexadecimal format," like that produced by the function **xdump()**. For example, the vertical hexadecimal dump of the string "hello world.\n" is:

```
0 hell o wo rld. .
  6666.6276.7662.0
  85CC.F07F.2C4E.A
```

The hexadecimal value of 'h' is 0x68, which appears vertically under the 'h'. The dump is broken into groups of four bytes; every unprintable character appears as '.'.

For details on **xdump()**, see the Lexicon entry for **libmisc**.

See Also

as, asfix, coff.h, commands, ld, libmisc

Notes

cdmp is an analogue of the UNIX command **cdump**.

cdplayer — Command

Play audio CDs
cdplayer [eject info pause play [*track***] resume skip stop volume** *level***]**

cdplayer gives you a text-based interface with which you can play audio compact disks (CDs) through a COHERENT CD-ROM device. It reads environmental variable **CD_DEVICE** for the name of the device to manipulate. If this variable is not set, by default **cdplayer** manipulates device **/dev/cdrom**.

cdplayer normally is invoked with one of the following commands. If you invoke it without a command (or with a command it does not recognize), it prints a usage message and exits. If an error occurs, **cdplayer** returns an exit status of one. **cdplayer** recognizes the following commands:

eject Eject the CD from the drive. Note that not every CD drive supports this feature; in particular, the Mitsumi model LU005 does not.

info Display information about the CD that is in the drive: the total number of tracks, total playing time, playing time per track, drive status, and track being played.

pause Pause the audio CD. Unlike the command **stop**, described below, **cdplayer** remembers the point at which playing stopped, and will resume playing at that point. If the CD is not playing, **cdplayer** ignores this command. To restart a paused CD, use the command **cdplayer resume**.

play [*track***]**
Play the CD, beginning at *track*. If no *track* is given, it begins as track one.

resume
Resume playing a paused CD. If the CD had not been paused, **cdplayer** ignores this command.

skip Skip to the next track. If the CD is on its last track, **cdplayer** returns it to its first track.

stop Stop playing this CD. If the CD is not being played, **cdplayer** ignores this command. The CD player "forgets" the point at which it had been playing the CD. To begin playing this CD again, use the command **cdplayer play**.

volume *level*
Set the CD drive's volume to *level*, which must be a number between 0 (softest) and 255 (loudest). Note that not every drive supports this feature.

Environment

CD_DEVICE — The CD-ROM device to manipulate.

See Also

CD-ROM, cdv, commands

Notes

cdplayer was written by Mark Buckaway (mark@datasoft.com) for the Linux operating system. Please direct comments concerning its COHERENT port to support@mwc.com. It is is distributed under the GNU Public License. Full source code for this program is available on the Mark Williams Bulletin Board and on other publically available systems.

cdrom.h — Header File

Definitions for CD-ROM drives
#include <sys/cdrom.h>

The header file **<sys/cdrom.h>** defines structures and IOCTLs used to manipulate CD-ROM drives.

See Also

CD-ROM, header files, ioctl

cdu31 — Device Driver

Driver for the Sony CD-ROM drives

cdu31 is a device driver for the Sony CD-ROM drive, models CDU31A and CDU33A. It has major-device number 14.

Normally, this device driver is included in the kernel when you install or update COHERENT. To configure this driver, log in as the superuser **root**, and execute script **/etc/conf/cdu31/mkdev**. Then run the command

```
/etc/conf/bin/idmkcoh -o coh.test
```

to build a test kernel that includes the driver.

Files

/dev/cdrom — Device applications read for CD-ROMs by default
/dev/rscd0 — Device for accessing CDU31A CD-ROM

See Also

CD-ROM, device drivers, hai

cdv — Command

Interface to CD-ROM devices
cdv [*directory***]**

The script **cdv** provides a easy-to-use interface to the set of commands that interrogate an ISO-9660 CD-ROM. It is designed to spare you the trouble of having to remember the names and syntax used by each of these commands. If you name a *directory* on its command line, **cdv** uses that directory within the CD-ROM's file system as its root file system; otherwise, it begins its work in the CD-ROM's default root directory. The advantage of this option is that CD-ROM file systems tend to hold many files, and reading the CD-ROM can be quite slow (depending upon the speed of your system and of your CD-ROM reader); making *directory* the root directory lessens the number of files **cdv** must paw through before it finds the material that interests you. Obviously, you must have some idea of the CD-ROM's contents before you can use this option.

After you invoke **cdv**, it displays the prompt:

```
Command:
```

Enter the command that you want **cdv** to execute, as follows:

cd *directory*
: Change directory. *directory* is the directory to enter. This can be a relative path name or absolute path name. As with the COHERENT command **cd**, you can use '.' and '..' as synonyms for, respectively, the current directory and the parent directory.

G *directory*
: Read the contents of *directory*.

g *file* Get *file*; copy it into the current directory.

N
n Because the contents of a CD-ROM's directory may not fit onto the screen, **cdv** lets you display a directory's contents one page at a time. These commands display the next page of the current directory's contents.

P
p Display the previous page of the current directory's contents.

Q
q Quit.

v *file* View *file*, which is on the CD-ROM. **cdv** displays *file* with the pager named in the environmental variable **$PAGER**. If this variable is not defined, it uses **more**.

! Invoke the shell. To return to **cdv**, type **exit**, to exit from the shell.

See Also

CD-ROM, cdview, commands

Notes

cdv was written by Chris Hilton.

cdview — Command

Read a file from a CD-ROM
cdview [*file***]**

The command **cdview** reads *file* from an ISO-9660 CD-ROM, and writes its contents to the standard output. If *file* names a directory on the CD-ROM, **cdview** writes its contents to the standard output.

cdview normally is used with the script **cdv**, which provides a kinder, gentler way to interrogate the device.

See Also

CD-ROM, cdv, commands

ceil() — Mathematics Function (libm)

Set numeric ceiling
#include <math.h>
double ceil(*z***) double** *z***;**

ceil() returns a double-precision floating-point number whose value is the smallest integer greater than or equal to *z*.

Example

The following example demonstrates how to use **ceil()**:

```
#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#define display(x) dodisplay((double)(x), #x)

dodisplay(value, name)
double value; char *name;
{
      if (errno)
            perror(name);
      else
            printf("%10g %s\n", value, name);
      errno = 0;
}
```

```
main()
{
      extern char *gets();
      double x;
      char string[64];

      for (;;) {
            printf("Enter number: ");
            if (gets(string) == NULL)
                  break;
            x = atof(string);

            display(x);
            display(ceil(x));
            display(floor(x));
            display(fabs(x));
      }
      putchar('\n');
}
```

See Also

abs(), fabs(), floor(), frexp(), libm
ANSI Standard §7.5.6.1
POSIX Standard, §8.1

cfgetispeed() — termios Macro (termios.h)

Get terminal input speed
#include <termios.h>
int cfgetispeed(*tty***)**
termios ******tty***;**

Macro **cfgetispeed()** returns the input speed of the terminal device. *tty* gives the address of a structure of type **termios**. It must have been initialized by a call to the **termios** routine **tcgetattr()**.

See Also

termios
POSIX Standard, §7.1.3

cfgetospeed() — termios Macro (termios.h)

Get terminal output speed
#include <termios.h>
int cfgetospeed(*tty***)**
termios ******tty***;**

Macro **cfgetospeed()** returns the input speed of the terminal device. *tty* gives the address of a structure of type **termios**. It must have been initialized by a call to the **termios** routine **tcgetattr()**.

See Also

termios
POSIX Standard, §7.1.3

cfsetispeed() — termios Macro (termios.h)

Set terminal input speed
#include <termios.h>
int cfsetispeed(*tty***,** *speed***)**
termios ******tty***;**
int *speed***;**

Macro **cfsetispeed()** sets the input speed of the terminal device.

tty gives the address of a structure of type **termios**. It must have been initialized by a call to the **termios** routine **tcgetattr()**. *speed* gives the speed to which the terminal device should be set. It must be one of the following constants:

B50	50 baud
B75	75 baud
B110	110 baud
B134	134.5 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud
B2400	2400 baud
B4800	4800 baud
B9600	9600 baud
B19200	19200 baud
B38400	38400 baud

You must call routine **tcsetattr()** for *tty* before this change can take effect.

See Also

termios
POSIX Standard, §7.1.3

cfsetospeed() — termios Macro (termios.h)

Set terminal output speed
#include <termios.h>
int cfsetospeed(*tty***,** *speed***)**
termios **tty***;*
int *speed***;**

Macro **cfsetospeed()** sets the output speed of the terminal device.

tty gives the address of a structure of type **termios**. It must have been initialized by a call to the **termios** routine **tcgetattr()**. *speed* gives the speed to which the terminal device should be set. It must be one of the following constants:

B50	50 baud
B75	75 baud
B110	110 baud
B134	134.5 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud
B2400	2400 baud
B4800	4800 baud
B9600	9600 baud
B19200	19200 baud
B38400	38400 baud

You must call routine **tcsetattr()** for *tty* before this change can take effect.

See Also

termios
POSIX Standard, §7.1.3

cgrep — Command

Pattern search for C source programs
cgrep [-clnsA] [-r *new***]** *expression file* **...**

cgrep is a string-search utility. It resembles its cousins **grep** and **egrep**, except that it is specially designed to be used with C source files. It checks all C identifiers against *expression* and prints all lines in which it finds a

match. **cgrep** allows you to search for a variable named 'i' without finding every 'if' and 'int' in your program. **cgrep** defines an "identifier" to be any variable name or C keyword. *expression* can be a regular expression; if it includes wildcard characters or '|'s, you must "quote it" to protect it against being modified by the shell. For details on the expressions that **cgrep** can recognize, see the Lexicon entry for **egrep**.

cgrep tests names that include the '.' and '->' operators against *expression*. Thus, to look for **ptr->val**, type:

```
cgrep "ptr->val" x.c
```

This finds **ptr->val** even if it contains spaces, comments, or is spread across lines. If it is spread across lines, it will be reported on the line that contains the last token. The only exception is if you include the **-A** option, in which case it will be reported on the line which contains the first token. This is to simplify MicroEMACS macros, as will be described below.

To find **structure.member**, type:

```
cgrep "structure\.member"
```

because '.' in a regular expression matches any character.

Do not include spaces in any pattern. Only identifiers and '.' or '->' between identifiers are included in the tokens checked for pattern-matching.

Command-line Options

cgrep recognizes the following command-line options:

-A Write all lines in which *expression* is found into a temporary file. Then, call MicroEMACS with its error option to process the source file, with the contents of the temporary file serving as an "error" list. This option resembles the **-A** option to the **cc** command, and lets you build a MicroEMACS script to make systematic changes to the source file. To exit MicroEMACS and prevent **cgrep** from searching further, **<ctrl-U> <ctrl-X> <ctrl-C>**.

-c Print all comments in each *file*. This form takes no expression.

-l List only the names of the files in which *expression* is found.

-n Prefix each line in which *expression* is found with its line number in the file.

-r Replace all expression matches with *new*. This option may not be used with any others, and it can only match simple tokens, not items like **ptr->val**. When **-r** is used and the input is **stdin**, a new file will always be created as **stdout**.

-s Print all strings in each *file*. This form takes no expression.

Examples

The command

```
cgrep tmp *.c
```

will find the variable name **tmp**, but not **tmpname**, or any occurrence of **tmp** in a string or comment.

The script

```
cgrep -c < myfile.c | wc -l
```

count the lines of comments in **myfile.c**.

The command

```
cgrep "x|abc|d" *.c
```

will find **x**, **ab**, or **d**. Note this is a regular expressions with a surrounding "^( )$" which is applied to every identifier. Thus, **reg*** will not match **register**, but **reg.*** will.

See Also

commands, egrep, grep, me

LEXICON

char — C Keyword

Data type

char is a C data type. It is the smallest addressable unit of data. According to the ANSI Standard, a **char** consists of exactly one byte of storage; a byte, in turn, must be composed of at least eight bits. **sizeof(char)** returns one by definition, with all other data types defined as multiples thereof. All Mark Williams compilers sign-extend **char** when it is cast to a larger data type.

Under COHERENT, a **char** by default is signed.

See Also

byte, C keywords, data formats, unsigned
ANSI Standard, §6.1.2.5

chase — Command

Highly amusing video game
/usr/games/chase [-c] [*speed* **]**

chase is a COHERENT version of a popular video game. It runs on the console with input from the console keyboard. **chase** assumes that the system console is a monochrome display adapter unless you select the **-c** color-display option.

To accomodate different computer system speeds and different levels of skill, **chase** prompts the user to type a speed when the game begins. Press **<return>** to try out the game with the default speed of ten; typing a higher number makes the game slower, a lower number makes it faster. If you can play at speed zero on a fast computer system, you play too many video games. If you know the speed you want, you can enter it as a command-line argument. If you see the boss coming, quit by pressing **<ctrl-C>**.

The Rules

The player (represented by a blinking shaded rectangle) attempts to evade four "ghosts" (represented by shaded rectangles with arrows) while erasing dots from the playing-board maze.

At the beginning of a game, the four ghosts are in the *ghost box* above the center of the maze and the player is below it. The maze is filled with dots, including four blinking diamonds called *power pellets*. The ghosts emerge from the ghost box and chase the player. The console arrow keys move the player left, right, up, or down through the maze. Typing '0' stops the player. The player continues to move in the same direction until a wall of the maze stops him, you type a '0', or you type another arrow key.

When the player eats a power pellet, he acquires super power and can chase the ghosts briefly; the ghosts change color while the player has super power. If the player catches a ghost, he scores a bonus and the ghost returns to the ghost box temporarily. Once a player eats all the dots on the board, the game continues at the next level.

The upper left corner of the screen displays a score and the current board level. Each dot the player eats scores ten points. The first ghost a player eats while he has super power scores 200 points, the second 400, the third 800, and the fourth 1,600. At certain times during the game, a bonus letter appears below the ghost box; the player scores 100 points for eating the bonus letter on level 'A', 300 on level 'B', 500 on level 'C', and so on.

The lower left corner of the screen displays the number of extra players remaining in the current game (initally two). Another bonus player appears every 10,000 points, to a maximum of three extra players. The game ends when the ghosts eat the last player.

See Also

commands

chdir() — System Call (libc)

Change working directory
#include <unistd.h>
chdir(*directory***) char** ***directory***;**

The *working directory* (or *current directory*) is the directory from which the search for a file name begins if a path name does not begin with '/'. By convention, the working directory has the name '.'. **chdir()** changes the working directory to the directory pointed to by *directory*. This change is in effect until the program exits or calls **chdir()** again.

See Also

cd, chmod(), chroot(), directory, libc, unistd.h
POSIX Standard, §5.2.1

Diagnostics

chdir() returns zero if successful. It returns -1 if an error occurred, e.g., that *directory* does not exist, is not a directory, or is not searchable.

check — Command

Check file system
check [-s] *filesystem ...*

check uses the commands **icheck** and **dcheck** to check the consistency of a file system. It acts on each argument *filesystem* in turn; it calls first **icheck** and then **dcheck** on each to detect problems.

If **-s** is specified, **check** attempts to repair any errors automatically. You should first unmount the file system, if possible. If the root device is involved, you should be in single-user mode and then reboot the system immediately (without typing **sync**).

See Also

clri, commands, icheck, ncheck, sync, umount

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.

This command has largely been superceded by **fsck**.

checkerr — Command

Check the mail system for errors
/usr/lib/mail/checkerr

The script **checkerr** reads error reports that have been deposited into the error directory **/usr/spool/smail/error**. If it finds an error, **checkerr** concatenates them into file **/usr/spool/smail/.checkerror**, and mails that file to user **postmaster** on your system. If mail cannot be sent to **postmaster** for any reason, **checkerr** leaves the file in place; when you next invoke this command, it will again try to mail the error messages.

See Also

commands, mail [overview], smail

checklist — System Administration

File systems to check when booting COHERENT
/etc/checklist

The file **/etc/checklist** names all COHERENT partitions on your hard disk. COHERENT executes **fsck** for each file named in this file. This ensures that the file-system of each partition is checked and cleaned before it is mounted.

When you add a new COHERENT partition to your system, you should insert its name (that is, the name of its raw device) into **/etc/checklist** to ensure that its file system is checked at boot time.

See Also

Administering COHERENT, brc

chgrp — Command

Change the group owner of a file
chgrp *group file ...*

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 identifier.

Only the superuser may use **chgrp**.

Files

/etc/group — Convert group name to group identifier

See Also

chmod, chmog, chown, commands

chmod — Command

Change the modes of a file
chmod +*modes file*
chmod -*modes file*

The COHERENT system assigns a *mode* to every file, to govern how users access the file. The mode grants or denies permission to read, write, or execute a file.

The mode grants permission separately to the owner of a file, to users from the owner's group, and to all other users. For a directory, execute permission grants or denies the right to search the directory, whereas write permission grants or denies the right to create and remove files.

In addition, the mode contains three bits that perform special tasks: the set-user-id bit, the set-group-id bit, and the save-text or "sticky" bit. See the Lexicon entry for the COHERENT system call **chmod()** for more information on how to use these bits.

The command **chmod** changes the permissions of each specified *file* according to the given *mode* argument. *mode* may be either an octal number or a symbolic mode. Only the owner of a *file* or the superuser may change a file's mode. Only the superuser may set the sticky bit.

A symbolic mode may have the following form. No spaces should separate the fields in the actual *mode* specification.

[which] how perm ... [, ...]

which specifies the permissions that are affected by the command. It may consist of one or more of the following:

a All permissions, equivalent to **gou**
g Group permissions
o Other permissions
u User permissions

If no *which* is given, **a** is assumed and **chmod** uses the file creation mask, as described in **umask**.

how specifies how the permissions will be changed. It can be

= Set permissions
+ Add permissions
- Take away permissions

perm specifies which permissions are changed. It may consist of one or more of the following:

g Current group permissions
o Current other permissions
r Read permission
s Setuid upon execution
t Save text (sticky bit)
u Current user permissions
w Write permission
x Execute permission

Multiple *how*/*perm* pairs have the same *which* applied to them. One or more specifications separated by commas tell **chmod** to apply each specification to the file successively.

An octal *mode* argument to **chmod** is obtained by *OR*ing the desired mode bits together. For a list of the recognized octal modes, see the Lexicon entry for **chmod()**.

Examples

The first example below sets the owner's permissions to read + write + execute, and the group and other permissions to read + execute. The second example adds execute permission for everyone.

```
chmod u=rwx,go=rx file
chmod +x file
```

See Also

chgrp, chmod(), chmog, chown, commands, ls, stat, umask

chmod() — System Call (libc)

Change file-protection modes
#include <sys/stat.h>
chmod(*file***,** *mode***)**
char ******file***; int** *mode***;**

chmod() sets the mode bits for *file*. The mode bits include protection bits, the set-user-id bit, and the sticky bit.

mode is constructed from the logical OR of the mode constants declared in the header file **stat.h**, as follows:

S_ISUID	Set user identifier on execution
S_ISGID	Set group identifier on execution
S_ISVTX	Save file on swap device ("sticky bit")
S_IRUSR	Read permission for owner
S_IWUSR	Write permission for owner
S_IXUSR	Execute permission for owner
S_IRGRP	Read permission for members of owner's group
S_IWGRP	Write permission for members of owner's group
S_IXGRP	Execute permission for members of owner's group
S_IROTH	Read permission for other users
S_IWOTH	Write permission for other users
S_IXOTH	Execute permission for other users

For directories, some protection bits have a different meaning: write permission means files may be created and removed, whereas execute permission means that the directory may be searched.

The save-text bit (or "sticky bit") is a flag to the system when it executes a shared for of a load module. After the system runs the program, it leaves shared segments on the swap device to speed subsequent reinvocation of the program. Setting this bit is restricted to the superuser (to control depletion of swap space which might result from overuse).

Only the owner of a file or the superuser may change its mode.

See Also

creat(), libc, stat.h
POSIX Standard, §5.6.4

Diagnostics

chmod() returns -1 for errors, such as *file* being nonexistent or the invoker being neither the owner nor the superuser.

chmog — Command

Change mode, owner, and group simultaneously
chmog *mod own grp file ...*

chmog combines the functionality of the commands **chmod**, **chown**, and **chgrp** into one command. This lets you fine-tune the permissions on *file*s without having to type three separate commands.

The arguments *mode*, *own*, and *grp* give, respectively, the mode, owner, and group to which **chmog** sets *file*. Setting any of these three arguments '-' means that that feature of *file* is not changed. For example, the command

```
chmog - bin bin file_name
```

changes the owner and group of file **file_name** to **bin** and does not alter **file_name**'s permissions.

For details on how to set *mode*, *own*, and *grp*, see the Lexicon entries for, respectively, **chmod**, **chown**, and **chgrp**.

See Also

chgrp, chmod, chown, commands

chown — Command

Change the owner of files
chown *owner file ...*

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, chmog, commands

chown() — System Call (libc)

Change ownership of a file
#include <unistd.h>
chown(*file, uid, gid***)**
char **file***file***; short** *uid, gid***;**

chown() changes the owner of *file* to user id *uid* and group id *gid.*

To change only the user id without changing the group id, use **stat()** to determine the value of *gid* to pass to **chown()**.

chown() is restricted to the superuser, because granting the ordinary user the ability to change the ownership of files might circumvent file space quotas or accounting based upon file ownership.

chown() returns -1 for errors, such as nonexistent *file* or the caller not being the superuser.

See Also

chmod(), libc, passwd, stat(), unistd.h
POSIX Standard, §5.6.5

chreq — Command

Change priority, lifetime, or printer for a job
chreq [-d*printer***] [-l***lifetime***] [-p***priority***]** *job*

The command **chreq** lets you change the printer, lifetime, and priority of a *job*, which identifies a print job spooled with the command **lp**. It recognizes the following options:

-d*printer* Move *job* to the queue for *printer*.

-l*lifetime* Change the lifetime of *job*, where *lifetime* is one of **T** (temporary), **S** (short-term), or **L** (long-term). Temporary lifetime means that a job "survives" in the spool directory for two hours after being spooled; short-term means that it survives 48 hours; and long-term that it survives for 72 hours. After a job's lifetime has expired, the print daemon **lpsched** removes it.

-p*priority* Change the despooling priority of *job* to *priority*, which is one of **0** (highest priority) to **9** (lowest priority). Jobs with high priority are printed before those with low priority. The default priority is **2**.

See Also

commands, lp, MLP_PRIORITY, printer

Notes

You can reset the default priority for print jobs by setting the environmental variable **MLP_PRIORITY**.

chreq is available only under COHERENT release 4.2 and subsequent releases.

chroot — Command

Change root directory
chroot *directory program ...*

The command **chroot** runs program *program* with root directory *directory*.

See Also

commands

Notes

Only the superuser **root** can use **chroot**.

chroot() — System Call (libc)

Change the root directory
#include <unistd.h>
int chroot(*path***)**
char **path***;*

The COHERENT system call **chroot()** changes the current process's root directory to that specified by *path*. Once the **chroot()** system call completes, all references to absolute directories (i.e., ones starting with '/') will actually refer to directory pointed to by *path*. It does not change the current directory.

chroot() is often used to add extra security to special or public login accounts.

See Also

chroot, libc

Notes

The process that invokes **chroot()** must be running as the superuser **root**, and *path* must name a valid directory.

chsize() — System Call (libc)

Change the size of a file
int chsize(*fd, size***);**
int *fd***; long** *size***;**

The COHERENT system call **chsize()** changes the size of the file associated with the file descriptor *fd* to be exactly *size* bytes long. If *size* is larger than the file's initial size, then **chsize()** pads the file with the appropriate number of extra bytes. If *size* is smaller than the initial size, then *chsize()* frees all allocated disk blocks between *size* and the initial size. The maximum file size as set by **ulimit()** is in force for calls to **chsize()**.

With a successful call, **chsize()** returns 0; otherwise, it returns -1 and sets **errno** to an appropriate value.

See Also

libc, open(), ulimit()

Notes

When you use **chsize()** to shorten a file, COHERENT frees all disk blocks beyond the new end-of-file mark. However, it does not zero out the bytes beyond the new end-of-file in the last allocated disk block. If you wish to obliterate a file, simply using **chsize()** to reset its size to zero will *not* do the trick.

When you use **chsize()** to lengthen a file, the new bytes beyond the initial size are simply those bytes that were in the final disk block beyond the original end-of-file marker. All additional bytes beyond that point are zeroes. The file system will not actually allocate new disk blocks to accomodate the new file size, but rather will create one or more sparse blocks.

The term *sparse block* refers to the fact that in the COHERENT file system, a disk block that would be all zeroes need not take up a physical disk block. Rather, COHERENT marks the i-node to indicate that the block is all zeroes, but does not allocate a physical block. This saves space on the disk.

A *sparse file*, is a file that contains one or more sparse blocks. The file system handles sparse files correctly; however, the command **fsck** may return the error message

```
Possible File Size Error
```

for them.

If you lengthen a file with **chsize()**, you may create a sparse file, which may in turn cause **fsck** to complain.

ckermit — Command

Interactive inter-system communication and file transfer
ckermit [-abcdefghiklpqrstwx] [*file ...* **]**

ckermit implements the **kermit** communications protocol. It lets you communicate with other systems via modem or network, and to exchange files with other systems that have also implemented the **kermit** protocol. Unlike the **kermit** command also included with the COHERENT system, **ckermit** uses an interactive shell to remove some of the pain from the process of exchanging files. The name **ckermit** relects the fact that this command is written in the C language, and so has been ported to many different machines and operating systems.

You can run **ckermit** in either *interactive mode* or *command mode*. Simply typing the command

```
ckermit
```

invokes **ckermit** in interactive mode: **ckermit** displays a prompt, waits for your command, executes, then prompts you for its next command. Typing the command line plus one or more arguments invokes **ckermit** in command mode: **ckermit** then reads the arguments from the command line and executes them. After execution of the commands, **ckermit** returns to interactive mode.

ckermit's command-line options name either actions or settings. An action option tells **ckermit** to send a file, receive a file, or connect to a remote system. The command line may contain no more than one action option. A settings option changes one or more of the internal values that control how **ckermit** operates; for example, one setting option lets you set the baud rate of the serial port that **ckermit** will be using. A command line can contain any number of settings options.

Command-Line Options

ckermit recognizes the following command-line options:

-a *filename* Give an alternate name to a file being transferred. For example, the command

```
ckermit -s foo -a bar
```

transmits the file **foo** to a remote system, but tells the remote system that the file is named **bar**. Likewise, the command

```
ckermit -ra baz
```

stores the first incoming file under the name **baz**.

If more than one file arrives or is sent, only the first file is affected by the **-a** option.

-b *baudrate* Set the baud rate of the device to *baudrate*.

-c Connect to serial port, and pass all subsequent typing to that port To resume talking to your local system, type the escape character followed by the letter 'c'. The escape character is set by default to **<ctrl-\>**, although you can change it if you wish.

-d Debug mode — record debugging information in the file **debug.log** in the current directory.

-e *n* Set the length of the packet to *n* where *n* is a number between ten and about 1,000. Lengths of 95 or greater require that the implementation of **kermit** on the remote system support the long-packet extension to the **kermit** protocol.

-f Send a "finish" command to a remote server.

-g *file* Ask a remote system to send *file* or *files*. The file name must use the remote system's own syntax; you must quote all characters normally expanded by the COHERENT shell, e.g.:

```
ckermit -g x\*.\?
```

-h Help — display a brief synopsis of the command-line options.

-i The "image" option: specify that the file being transmitted or received is an eight-bit binary file, and therefore no conversion should be performed upon the data being received.

-k Passively receive file or files, copying them to standard output.

-l *device* Name the serial device to be used. For example

```
ckermit -l /dev/com2l
```

tells **ckermit** to use device **/dev/com2l**.

-n Like **-c,** but used after a protocol transaction has occurred. You can use both **-c** and **-n** in the same command.

-p *x* Set parity, where *x* is one of **e**, **o**, **m**, **s**, or **n** (respectively, even, odd, mark, space, or none). If parity is other than none, then **ckermit** uses the eighth-bit prefixing mechanism to transfer binary data, provided the impementation of **kermit** on the remote system agrees. The default parity is none.

-q Quiet — suppress screen update during file transfer; for example, this lets you transfer a file in the background.

-r Receive a file or files. Wait passively for files to arrive.

-s *file* Send the specified *file* or *files*. If *fn* is '-' then **ckermit** sends from standard input, which may come from a file:

```
ckermit -s - < foo.bar
```

or come from a parallel process:

```
ls -l | ckermit -s -
```

You cannot use this mechanism to send text typed from the keyboard. To send a file named '-', precede it with a path name, e.g.:

```
ckermit -s ./-
```

-t Specify half duplex, line turnaround with XON as the handshake character.

-w Write-Protect — avoid file-name collisions for incoming files.

-x Begin server operation. This option can be used in either local or remote mode.

If **ckermit** is in local mode, shows the progress of the file transfer. A dot is printed for every four data packets; other packets are shown by type (e.g., 'S' for Send-Init); 'T' is printed when there's a timeout; and '%' is printed for each retransmission.

During file transfer, you can type the following "interrupt" commands:

<ctrl-F> Interrupt the current file and go on to the next, if any.

<ctrl-B> Interrupt the entire batch of files and terminate the transaction.

<ctrl-R> Resend the current packet.

<ctrl-A> Display a status report for the current transaction.

These interrupt characters differ from the ones used in other implementations of **ckermit** to avoid conflict with the COHERENT shell's interrupt characters.

Interactive Operation

When you invoke **ckermit** in interactive mode, it displays the following prompt.

```
C-Kermit>
```

Type any valid **ckermit** command; the set of valid commands is described below. **ckermit** executes the command and then prompts you for another. The process continues until you tell it to quit.

Commands begin with a keyword, normally an English verb, such as **send**. You can abbreviate any keyword, as long as you type enough characters to distinguish it from all other keywords. Certain commonly used keywords (e.g., **send**, **receive**, **connect**) have special non-unique abbreviations (respectively, 's', 'r', and 'c').

Certain characters have special functions in interactive commands:

? Print a message that explains what is possible or expected at the current point within a command. Depending upon the context, the message may be a brief phrase, a menu of keywords, or a list of files.

<esc> Request completion of the current keyword or file name, or insertion of a default value. **ckermit** will beep if the requested operation fails. **<tab>** does the same thing.

<del> Delete the previous character from the command. **<backspace>** does the same thing.

<ctrl-W> Erase the rightmost word from the command line.

<ctrl-U> Erase the entire command.

<ctrl-R> Redisplay the current command.

<space> Delimit fields (keywords, filenames, numbers) within a command.

<return> Execute the command.

**** Insert any of the above characters into the command, literally. To enter a literal backslash, type two backslashes in a row (\\). Typing one backslash immediately **<return>** lets you continue the command on the next line.

ckermit recognizes the following interactive commands:

! *command* Execute a shell command. A space must follow the **!**.

% A comment. **ckermit** ignores everything that follows the **%**.

bye Terminate and log out a remote **kermit** server.

close Close a log file.

connect Connect to the remote system.

cwd *directory*
Change the working directory to *directory*.

dial Dial a telephone number.

directory Display a directory listing.

echo Display arguments literally. Useful in take-command files.

exit Exit from the program, closing any open logs.

finish Instruct a remote **kermit** server to exit, but not log out.

get Get files from a remote **kermit** server.

hangup Hang up the telephone.

help Display a help message for a given command.

log Open a log file — debugging, packet, session, transaction.

quit Same as **exit**.

receive Passively wait for files to arrive.

remote Issue file-management commands to a remote **kermit** server.

script Execute a login script with a remote system.

send *file* Send *file* to the remote **kermit** server.

server Begin server operation.

set Set various internal parameters.

show Display values of parameters, program version, etc.

space Display current disk space usage.

statistics Display statistics about most recent transaction.

take Execute commands from a file.

Interactive **ckermit** accepts commands from files as well as from the keyboard. Upon startup, **ckermit** looks for the file **.kermrc** first in directory **$HOME** and then in the current directory; if it finds the file, it executes all commands it finds therein. These commands must be in interactive format. Command files may be nested to any reasonable depth.

The set Command

As noted above, the **set** command lets you set the internal parameters by which **ckermit** operates. The **set** command recognizes the following arguments:

block-check
Level of packet error detection.

delay Time to wait before sending first packet.

duplex Specify which side echoes during connect mode.

escape-character
Character to prefix *escape commands* during connect mode.

file Set various file parameters.

flow-control
Communication line full-duplex flow control.

handshake Communication line half-duplex turnaround character.

line Communication-line device name.

modem-dialer
Type of modem-dialer on communication line.

parity Communication line character parity.

prompt Change the **ckermit** program's prompt.

receive Set various parameters for inbound packets.

retry Set the packet retransmission limit.

send Set various parameters for outbound packets.

speed Communication line speed.

Remote Commands

ckermit also has a suite of commands that are sent to the remote system for execution. They are as follows:

cwd Change remote working directory (also, **remote cd**).

delete Delete remote files.

directory Display a listing of remote file names.

help Request help from a remote server.

host Issue a command to the remote host in its own command language.

space Display current disk space usage on remote system.

type Display a remote file on your screen.

who Display the users logged in to the remote system, or get information about a user.

Files

.kermrc — **ckermit** initialization commands

See Also

commands, kermit, uucp

LEXICON

Notes

The **kermit** protocol was developed at the Columbia University Center for Computing Activities. **ckermit** is copyright © by the Trustees of Columbia University.

On some remote systems, the command **hangup** does not hang up the telephone properly. If this occurs, add the following macro to file **$HOME/.kermrc**:

```
define myhangup sleep 2,output +++,sleep 2,output ATH0\13
```

This create a macro named **myhangup**, which you can invoke to hang up the remote telephone. To test the proper load of the macro, type the following at the **ckermit** prompt:

```
show macro myhangup
```

It should show the command sequence. If it is intact, you can execute the new **hangup** command by typing **myhangup**.

Please note that **ckermit** is provided in binary form per the licensing terms set forth by its copyright holders. It is distributed as a service to COHERENT customers, as is. It is not supported by Mark Williams Company. *Caveat utilitor.*

clear — Command

Clear the screen
clear

The command **clear** reads the **termcap** description of your terminal and uses the information therein to clear your terminal's screen. The environmental variable **TERM** must define your terminal's type.

See Also

commands, TERM, termcap

clearerr() — STDIO Function (libc)

Present stream status
#include <stdio.h>
clearerr(*fp***) FILE** ***fp*;**

clearerr() resets the error flag of the argument *fp*. If an error condition is detected by the related macro **ferror**, **clearerr()** can be called to clear it.

Example

For an example of this function, see the entry for **ferror()**.

See Also

ferror(), libc
ANSI Standard, §7.9.10.1
POSIX Standard, §8.1

clist.h — Header File

Character-list structures
#include <sys/clist.h>

The header file **clist.h** holds definitions useful to functions that manipulate character lists. It defines the character-list structure **CLIST** and the character-queue structure **CQUEUE**.

See Also

header files

clock — Device Driver

Read the system clock
/dev/clock

The file **/dev/clock** lets you read and set your system's clock. It is a part of the driver **mem**, which manages memory; thus, it has major number 0 and minor number 5.

The real time clock occupies the first 14 bytes of nonvolatile RAM (**/dev/cmos**). The difference between

/dev/cmos and **/dev/clock** is that the latter device locks the circuit during a read, so that the clock will not be updated as it is being read.

/dev/clock limits access to a 14-byte data area. Attempts to read or write beyond this limit will fail. **/dev/clock** stores the system time in binary-coded decimal (BCD). For details on BCD, see the Lexicon entry for **float**.

The COHERENT command **ATclock** reads this device and writes to it.

See Also

ATclock, cmos, device drivers, float

clock() — Time Function (libc)

Get processor time
#include <time.h>
clock_t clock();

The function **clock()** calculates and returns the amount of processor time a program has taken to execute to the current point. Execution time is calculated from the time the program was invoked. This, in turn, is set as a point from the beginning of an era that is defined by the implementation. Under COHERENT, time is recorded as the number of milliseconds since January 1, 1970, 0h00m00s GMT.

The value **clock()** returns is of type **clock_t**, which is defined in header file. **time.h**. If **clock()** cannot determine execution time, it returns -1 cast to **clock_t**.

To calculate the execution time in seconds, divide the value returned by **clock()** by the value of the macro **CLK_TCK**, which is also defined in **time.h**.

Example

This example measures the number of times a **for** loop can run in one second on your system. This is approximate because **CLK_TCK** can be a real number, and because the program probably will not start at an exact tick boundary.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

main()
{
      clock_t finish;
      long i;

      /* finish = about 1 second from now */
      finish = clock() + CLK_TCK;
      for(i = 0; finish > clock(); i++)
            ;

      printf("The for() loop ran %ld times in one second.\n", i);
      return(EXIT_SUCCESS);
}
```

See Also

difftime(), libc, mktime(), time.h
ANSI Standard, §.12.2.1

close() — System Call (libc)

Close a file
#include <unistd.h>
int close(*fd***) int** *fd***;**

close() closes the file identified by the file descriptor *fd*, which was returned by **creat()**, **dup()**, **open()**, or **pipe()**. **close()** also frees the associated file descriptor.

Because each program can have only a limited number of files open at any given time, programs that process many files should **close()** files whenever possible. The function **exit()** automatically calls **fclose()** for all open files; however, the system call **_exit()** does not.

Example

For an example of this function, see the entry for **open()**.

See Also

creat(), libc, open(), unistd.h
ANSI Standard, §4.9.3
POSIX Standard, §6.3.1

Diagnostics

close() returns -1 if an error occurs, such as its being handed a bad file descriptor; otherwise, it returns zero.

closedir() — General Function (libc)

Close a directory stream
#include <dirent.h>
int closedir(*dirp***)**
DIR ******dirp***;**

The COHERENT function **closedir()** is one of a set of COHERENT routines that manipulate directories in a device-independent manner. It closes the directory stream pointed to by *dirp*.

closedir() returns zero if no error occurs. If something goes wrong, it returns -1 and sets **errno** to an appropriate value.

Example

For an example of this system call, see the Lexicon entry for **opendir()**.

See Also

dirent.h, getdents(), libc, opendir(), readdir(), rewinddir(), seekdir(), telldir()
POSIX Standard, §5.1.2

Notes

The COHERENT implementation of the **dirent** routines was written by D. Gwynn.

clri — Command

Clear i-node
/etc/clri *filesystem inumber ...*

clri zeroes out each i-node with *inumber* on *filesystem*. *filesystem* is almost always a device-special file that corresponds to a disk device, e.g., **/dev/rat0a** or **/dev/rsd1c**. The raw device should be used. For example, the command

```
/etc/clri /dev/rat0a 8250
```

clears i-node 8250 on the file system on device **/dev/rat0a**, which is the first partition on your first AT hard disk.

The user must have read and write permission on the *filesystem*. If the file that *inumber* identifies is open, then **clri** probably will not work as you expect: the system maintains in core memory a copy of all active i-nodes, and the kernel will eventually write this copy to disk, thus undoing the action of **clri**. To ensure that this does not happen, unmount the file system before you running **clri**. If the i-node is for the root file system, reboot the system immediately after you run **clri**.

See Also

commands, dcheck, fsck, icheck, i-node, umount

cmos — Device Driver

Device for reading CMOS

The file **/dev/cmos** the entry via which you can read your system's CMOS. It is a part of the driver **mem**, which manages memory; thus, it has major number 0 and minor number 3.

The CMOS is a special, non-volatile area of random-access memory (RAM) that holds information about your system's configuration. The following gives the common meanings assigned to the various byte positions within the CMOS area:

Real-time clock:

0x00	Seconds
0x01	Alarm, seconds
0x02	Minutes
0x03	Alarm, minutes
0x04	Hours
0x05	Alarm, hours
0x06	Day of the week
0x07	Day of the month
0x08	Month
0x09	Year
0x0A	Update in progress

Diagnostic power byte:

0x0E
Bit 7 — Chip lost power
Bit 6 — Bad checksum
Bit 5 — Bad configuration byte
Bit 4 — Bad memory size
Bit 3 — Bad hard-disk byte
Bit 2 — Bad time of day

Restart-status byte:

0x0F Reloaded when restarting, e.g., returning from protected mode

Floppy-disk drive, drives A and B:

0x10
Bits 7-4 — Drive A:
0 = no drive
1 = 360-kilobyte drive
2 = 1.2-megabyte drive
3 = 720-kilobyte drive
4 = 1.44-megabyte drive
Bits 3-0 — Drive B:
0 = no drive
1 = 360-kilobyte drive
2 = 1.2-megabyte drive
3 = 720-kilobyte drive
4 = 1.44-megabyte drive

Floppy-disk drive, drives C and D:

0x11
Bits 7-4 — Drive C:
0 = no drive
1 = 360-kilobyte drive
2 = 1.2-megabyte drive
3 = 720-kilobyte drive
4 = 1.44-megabyte drive
Bits 3-0 — Drive D:
0 = no drive
1 = 360-kilobyte drive
2 = 1.2-megabyte drive
3 = 720-kilobyte drive
4 = 1.44-megabyte drive

Hard-disk drive:

0x12
Bits 7-4 — First hard-disk drive
0 = No drive
1-3 = Type 1-15
F = Use contents of byte 19
Bits 3-0 — Second hard-disk drive
0 = No drive
1-3 = Type 1-15
F = Use contents of byte 1A

Configuration of equipment:

0x014 Bits 7-6 — Floppy disks
00 = one floppy-disk drive
01 = two floppy-disk drives
10 = three floppy-disk drives
11 = four floppy-disk drives
Bits 5-4 — Type of display
00 = EGA/VGA
01 = CGA 40×25
10 = CGA 80×25
11 = monochrome display
Bit 1 — floating-point coprocessor installed
Bit 0 — Floppy-disk drive present

Memory:

0x15-0x16 Amount of memory below one megabyte
0x17-0x18 Amount of memory above one megabyte

Type of hard disk:

0x19 Type of first hard disk. Read only when bits 7-4 of byte 0x12 equal 0xF.
0x21 Type of second hard disk. Read only when bits 3-0 of byte 0x12 equal 0xF.

Miscellaneous:

0x2E-0x2F Checksum for bytes 0x10 through 0x2D
0x30-0x31 Indicate memory size above one megabyte
0x32 Century byte (BCD)
0x33 Flag for power-on information:
Bit 7 — Top 128 kilobytes of RAM is installed (shadow RAM is available)
Bit 6 — First boot after running set-up routine

/dev/cmos limits access to a 256-byte data area. Any attempt to read or write beyond this limit will fail.

See Also

ATclock, clock, device drivers, RAM

Notes

If you want to read or set the real time clock, then you should use **/dev/clock** instead of **/dev/cmos**.

Vendor-specific information, e.g., your system's memory configuration, is often kept in the CMOS area at locations beyond those documented above. Therefore, writing to undocumented regions of the CMOS area is extremely unwise: your computer could subsequently refuse to boot up properly. *Caveat utilitor.*

cmp — Command

Compare bytes of two files
cmp [-ls] *file1 file2* **[***skip1 skip2***]**

The command **cmp** compares two files byte by byte for equality. *file1* and *file2* name the files to compare; the file name '-' indicates the standard input.

If **cmp** finds two bytes that differ, it prints the number of the byte at which the discrepancy occurs, then exits. If it encounters EOF on one file but not on the other, it prints the message:

```
EOF on filen
```

cmp recognizes the following command-line options:

-l Note each differing byte by printing the positions and octal values of the bytes of each file.

-s Print nothing, but return the exit status.

By default, **cmp** begins at byte 1 of each file. The optional arguments *skip1* and *skip2* are integer values that tell **cmp** to skip that many bytes for the corresponding file before it begins the comparison. For example, the command

```
cmp FOO BAR 35 40
```

tells **cmp** to skip the first 35 bytes of **FOO** and the first 40 bytes of **BAR** before it begins to compare them.

See Also

commands, diff, sh, zcmp

Diagnostics

cmd returns zero for identical files, one for non-identical files, and two for errors, e.g., bad command or inaccessible file.

coff.h — Header File

Format for COFF objects
#include <coff.h>

coff.h describes the Common Object File Format (COFF), which is the object format used by COHERENT 386.

What Is COFF?

In brief, COFF is the UNIX System V standard for file formats. It defines the formats for relocatable object modules, for executable files, and for archives.

A COFF file is built around three sections, or *segments*:

text This holds executable machine code. It is write protected — the operating system is forbidden to overwrite it. (This is why operating systems that use COFF or similar formats are said to run in "protected mode.")

data This holds initialized data, that is, the data that the program finds when it begins execution. The program can read and write into this segment.

bss This segment holds unitialized data. It is simply a mass of space that is initialized to zeroes. It is contiguous with the **data** segment. The term **bss** from the old IBM mainframe days, and stands for "block started by symbol".

Not all segments have to be included in every COFF file. Further, some implementations of COFF define their own segments that manipulate special features of the operating system or hardware.

The following describes the structure of a COFF file. The areas within the file are described in the order in which they appear.

1. *file header*
This holds information set when the file was created, such as the date and time it was created, the number of segments in the file, a pointer to the symbol table, and status flags.

2. *optional header*
This gives information set at run-time, such as the address of the program entry point, and the size of the code and data segments.

3. *segment headers*
The next area holds a header for each segment in the file. Each header describes its segment's characteristics and contains pointers to the segment's contents, relocation information, line-number information, and other useful addresses.

4. *segment contents*
The next area holds the contents of the segments used in this file.

5. *relocation information*
The fifth area gives relocation information, one set of information for each segment in the file. The linker **ld** uses this information to generate the executable file at link time.

6. *line-number information*
This area holds debug information, one set of information for each segment. This area is optional.

7. *symbol table*
This area holds information used by both the linker and the debugger.

8. *string table*
This table holds very long names of variables.

Most of this information is irrelevant to the average user, or even the average developer of software. To the average user, COFF is "a machine that would go of itself"; you can run or compile programs without worrying what the linker puts where, or why. These details, however, can be very important if you are writing tools that manipulate the internals of files, such as archivers or debuggers. If you need detailed information on COFF and how to manipulate it, see *Understanding and Using COFF* (citation appears below).

For more information on how the COFF format affects COHERENT's language tools, see the Lexicon articles for **ar**, **as**, **cc**, **db**, and **ld**.

See Also

ar, as, cc, cdmp, coffnlist(), file formats, header files, ld
Gircys, G.R.: *Understanding and Using COFF*. Sebastopol, Calif., O'Reilly & Associates, Inc., 1988.

coffnlist() — General Function (libc)

Symbol table lookup, COFF format
#include <coff.h>
coffnlist(*fn***,** *nlp***,** *names***,** *count***)**
char **fn***fn***;**
SYMENT **nlp***nlp***;**
char **names***names***;**
int *count***;**

The function **coffnlist()** finds one or more names in the symbol table of a file in the COFF format.

You must arrange the names you seek into the form of a COFF symbol table. All long names (i.e., names longer than four characters) must be strung together like the COFF long-symbol-name section. Give each name an **n_type** of -1. After the call, any unfound names will still have this **n_type**, as a sign that it could not be found. Thus, you can use the same table to search several different COFF files.

fn points to the name of the file to be searched. *nlp* points to an array of type **SYMENT**. This structure is defined in header file **coff.h** as follows:

```
typedef     struct      syment      {
      union {
            char _n_name[SYMNMLEN]; /* Name */
            struct {
                  long _n_zeroes; /* If name[0-3] zero, */
                  long _n_offset; /* string table offset */
            } _n_n;
            char *_n_nptr[2];
      } _n;
      long        n_value;    /* Value */
      short       n_scnum;    /* Section number */
      unsigned short n_type;  /* Type */
      char        n_sclass;   /* Storage class */
      char        n_numaux;   /* Auxilliary entries */
#pragma align 2
} SYMENT;
#pragma align
```

count gives the number of symbols being sought. If there are long names, the displacement works from the *names* parameter.

Each item being sought must have 0xFFFF in its **n_type** field. This allows **coffnlist()** to be used on several files in order.

coffnlist() opens and reads the file pointed to by *fn*. It then scans the symbol table and tries to find a symbol with an **n_type** of 0xFFFF. Upon finding this entry, **coffnlist()** fills in the fields of the symbol entry.

coffnlist() returns zero if anything goes wrong, such as an inability to open the file *fn*. Otherwise, it returns one.

Example

The following example looks up three symbol names in the symbol table of file **tx.o**.

```
#include <stdio.h>
#include <coff.h>

main()
 {
      int i;
      static SYMENT sym[3];   /* the table of names to find */

      /* the long names section */
      static char long_names[] = "a_very_long_name";
      strcpy(sym[0].n_name, "x");   /* look up x */
      sym[0].n_type = -1;

      strcpy(sym[1].n_name, "y");   /* look up y */
      sym[1].n_type = -1;

      sym[2].n_zeroes = 0;    /* look up a_very_long_name */
      /* the long name table starts with a long giving its length
        * offsets are from the beginning of that long. Therefore
       * the n_offset of the first field is 4 not zero */
      sym[2].n_offset = sizeof(long);
      sym[2].n_type = -1;

      /* do lookups */
      if (!coffnlist("tx.o", sym, long_names, 3))
            exit(1);

      /* show off results */
      for (i = 0; i < 3; i++) {
            if (sym[i].n_type != -1)
                  printf("%s found at %x\n",
                        (sym[i].n_zeroes ? sym[i].n_name :
                              long_names + sym[i].n_offset - sizeof(long)),
                              sym[i].n_value);
      }
}
```

See Also

coff.h, libc, nlist()

coh_intro — Command

Tour the COHERENT file system
/etc/coh_intro [> *outfile* **]**

The command **coh_intro** walks you through the COHERENT file system. It gives you a brief introduction to each directory in the root file system, describes what it holds, and displays its contents.

This command is designed chiefly for a newcomer to COHERENT, to help teach her about the structure and operation of the COHERENT file system. An experience user may also wish to run **coh_intro** from time to time, in order to take a snapshot of her systems' current configuration.

See Also

commands

coherent.h — Header File

Miscellaneous useful definitions
#include <sys/coherent.h>

The header file **coherent.h** defines various useful types and objects. Among other things, it defines the structure **TIME**.

See Also

header files

COHERENT — Summary

Principles of the COHERENT System

This article describes COHERENT: its features, properties, and what sets it apart from other operating systems. It also gives tips on how to port an application to COHERENT, and describes how to un-install COHERENT from your system. For information on how COHERENT compares with MS-DOS, see the Lexicon article on **MS-DOS**.

What Is COHERENT?

COHERENT is a multiuser, multitasking operating system. *Multiuser* means that with COHERENT, more than one person can use your computer at any given time. *Multitasking* means that with COHERENT, any user can run more than one program at any given time. The design of COHERENT employs a few elegant concepts to give you a powerful and flexible system that is easy to use.

What is an Operating System?

An *operating system* is the master program that controls the operation of all other programs. It loads programs into memory, controls their execution, and controls a program's access to peripheral devices, such as printers, modems, and terminals.

Some operating systems permit only one user to use the computer at a time; and that user can run only one program at a time. However, you may well want your computer to support more than one user at a time, and run more than one program at a time. Sharing not only yields many economies (such as allowing a group of users to share one printer), but also allows the users to communicate with each other and so work together more efficiently.

Any multitasking operating system must be able to do the following tasks efficiently:

- Schedule computer time
- Control mass-storage devices (disks and tape drives)
- Organize disk-storage space
- Protect programs from conflict
- Protect stored information from destruction
- Ease cooperation among users

Today's operating systems also provide *tools*. These are programs that are bundled with the operating system, and that are designed to help you do your work more efficiently. For example, you need editors, compilers, debuggers, and assemblers to develop and test programs. Text formatters and spelling checkers help you write memoranda, manuals, or books. Command processors (also called *shells*) help you run the computer easily. Status checkers tell you what programs are being run, who is using the system, and how much space is left on your disk.

The combination of operating system and its tools transforms a boxful of wires and circuits into a useful machine.

COHERENT Documentation

This manual is designed to walk you through the COHERENT system. It consists of two parts: *tutorials* and *Lexicon*.

Each tutorial introduces a particular aspect of COHERENT. If you are a beginner, you should read the tutorials *Using the COHERENT System*, *Introducting*sh,*the*Bourne*Shell*, and *Introduction to MicroEMACS*. These will give you the basic information and basic skills you need to run COHERENT efficiently. A beginner who is interested in learning about the C language should look at the tutorial *The C Language*.

The tutorial *The make Programming Discipline* introduces the tool **make**. This tool is essential to building any complex tool under COHERENT. If you are going to be building tools under COHERENT, you should look at this tutorial.

The tutorial *UUCP, Remote Communications Utility* introduces UUCP. This bundle of programs lets your computer exchange mail and files with other computers, even if it is unattended. If you are all interested in networking with other computers (or plugging into the Internet), you should look at this tutorial.

The other tutorials introduce tools that are interest to advanced users.

The Lexicon fills the latter two thirds of this manual. It consists of more than 1,000 articles. The articles are printed in alphabetical order, to make it easy for you to find the one you want.

Most articles discuss a single aspect of the COHERENT system. Some articles, called "overview" articles, give a

broader discussion of an entire topic. Three overview articles are of particular interest:

Using COHERENT
: This article discusses the parts of COHERENT that are of interest to an ordinary user. It describes such matters as the commands available with COHERENT, and how a user can manage his own account.

Programming COHERENT
: This introduces the programming tools available under COHERENT; points to where you can find information about the COHERENT implementation of the C programming language; and points to where you can find information about the library routines and system calls that you can use in a program.

Administering COHERENT
: This article discusses how to administer COHERENT. It points to where you can find information on how to connect peripheral devices; manage mail and UUCP; change some of COHERENT's default behaviors; and modify and rebuild the COHERENT kernel. It also points to the articles that describe the files with which COHERENT manages itself.

If you cannot easily find an article that gives you the information you want, look in the index in the back of the manual. There is a good chance that you will find an entry there that points to the information you need. Also, you can use the command **apropos** to search the on-line version of the Lexicon for a key word that interests you. For details on this command, see its entry in the Lexicon.

How To Un-install COHERENT

To remove (or "un-install") COHERENT from your system, do the following:

1. Log in as the superuser **root**.
2. Invoke the COHERENT version of **fdisk**.
3. Choose the option to change all logical partitions. Don't change *any* parameters of any MS-DOS partitions.
4. Change *all* COHERENT partitions to type **Unused** with a size of 0, starting and ending at 0.
5. Exit **fdisk** and update the partition table.
6. Reboot the computer and run the MS-DOS **fdisk** utility to create a new MS-DOS partition table. Turn the unused space (formerly the COHERENT partitions) into an MS-DOS EXT partition. If you already have an MS-DOS EXT partition, change its parameters so that it incorporates the unused space.
7. Create one or more logical drives in the MS-DOS EXT partition.
8. Format the new logical drives using the MS-DOS **format** command.

Repeated tests with MS-DOS have shown that the above directions work. However, given the many flavors and releases of MS-DOS in circulation, Mark Williams Company cannot guarantee that the above steps will always work with MS-DOS. If they do not, consult your MS-DOS manual for creating a DOS partition table and file system on a new hard drive. If that information is not available, telephone Microsoft Technical Support at (206)454-2030.

Uninstalling the Mark Williams Bootstrap

The following describes how to remove the Mark Williams bootstrap program. You must do this if you are un-installing COHERENT from your system.

To remove the Mark Williams master boot program, you must overwrite the master boot-block on hard drive 0 with another boot program. Usually, this is the MS-DOS master boot program. Beginning with release 5.0, the MS-DOS version of **fdisk** has the switch **/mbr** that builds a new bootstrap program. All versions of the MS-DOS edition of **fdisk** writes a new master boot program if no valid signature appears at the end of the current contents of the master-boot block.

If you have MS-DOS version 5.0 or later, simply boot MS-DOS and run the command:

```
fdisk /mbr
```

If your version of MS-DOS predates release 5.0, you must modify the last two bytes of the master-boot block (to remove the magic "signature" that indicates a valid bootstrap program) then boot MS-DOS and run its version of **fdisk**.

Warning: See the note in the preceeding section about MS-DOS **fdisk** — back up your hard drive is backed up before you try this! There are several ways by which you can invalidate the signature at the end of the master-boot block. One way is to copy any sort of garbage into the master-boot block. You can (1) reformat cylinder 0 of your

hard drive — for example, using the **DIAGNOSTICS** menu of the AMI BIOS — or (2) use COHERENT to overwrite the block, e.g., with the command:

```
dd if=/coherent of=/dev/at0x count=1
```

The master-boot block is the first physical sector of the hard drive, i.e., cylinder 0, head 0, sector 1. (Note that numbering of sectors begins with one, not zero.) The MWC master bootstrap is part of the initial program load, and does not belong to any operating system because it runs before an operating system is loaded.

Please read the following carefully before you attempt erase the master-boot block:

Mark Williams Company can make no promises or guarantees concerning the behavior of any given version of the MS-DOS **fdisk**. Every version of the MS-DOS **fdisk** that we have tested does not recognize partitions allocated for other operating systems: MS-DOS cannot delete, or even display, such partitions. Certain configurations of empty partitions cause MS-DOS **fdisk** to hang.

Worst of all, don't expect *any* data on your hard drive to be available after MS-DOS **fdisk** rewrites an invalid master-boot block. Our experience is that MS-DOS **fdisk** erases all data in all partitions, even if previously existing MS-DOS partitions are re-allocated with identical cylinder ranges as at the time of their initial creation. *Caveat utilitor!*

cohtune — Command

Set a variable within a device driver
cohtune *driver tagfield* "*tagfield* **=** *value*"

The command **cohtune** sets the *tagfield* to *value* within the given device driver *driver*. You can then use the command **idmkcoh** to build a new kernel that incorporates your changes. When you boot the new kernel, your changes will have been made.

cohtune works by modifying the file **Space.c** for *driver*. Each device driver has such a file, that sets user-definable dimensions of its operation. When you invoke the command **idmkcoh** to build a new kernel, COHERENT automatically checks whether a **Space.c** module that have changed, compiles it, and links it into the newly built kernel. **idmkcoh** also recompiles every **Space.c** whenever you change a tunable variable in the kernel, just to ensure that all drivers are synchronized with changes in the kernel.

For example, the file **/etc/conf/hai/Space.c** gives the user-settable variable for the driver **hai**, which is COHERENT's host-adapter-independent SCSI driver. This file contains, among others, the variable **HAI_TAPE**. This variable is a bit-map; bit *n* is turned on if there is a SCSI tape device at SCSI ID *n*. If you have installed a SCSI tape as SCSI device 3, then type the following command:

```
cohtune hai "HAI_TAPE" "int HAI_TAPE = 0x08"
```

The value 0x08 turns on bit 3. As you can see, **cohtune** finds the line in **/etc/conf/hai/Space.c** that contains the string **HAI_TAPE** and is not commented out of the source, and replaces it with the line

```
int HAI_TAPE = 0x08
```

You can read a driver's **Space.c** to see how you can configure it. **Space.c** also gives some useful clues as to how the driver works and how it is currently configured.

You should *never* modify a **Space.c** by hand. If you do so, you run the risk of building a kernel that does not boot, or trashes your file system.

See Also

commands, device drivers, idenable, idmkcoh, idtune

Notes

cohtune cannot be used with STREAMS drivers.

Note that **cohtune** peforms no checks whatsoever on the content of the string with which you replace *tagfield* It should only be used by people familiar with C programming, because setting invalid values may cause errors that are difficult to diagnose. *Caveat utilitor*.

Because of the primitive nature of **cohtune**, we recommend that users not use it directly, but work instead through the configuration shell scripts supplied by the driver's developer (which typically live in directory **/etc/conf/***driver*) that can interactively generate the correct sequence of **cohtune** commands.

col — Command

Remove reverse and half-line motions
col [**-bdfx**][**-p***n*]

The command **col** reads 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. It removes **<esc>** (ASCII 033) from the input stream if it is 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.

col recognizes the following 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

ASCII, commands, nroff

Notes

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

comm — Command

Print common lines
comm [-123] *file1 file2*

The command **comm** prints the lines unique to *file1* in the first column, the lines unique to *file2* in the second column, and the lines common to both in the third. Both *file1* and *file2* should be sorted in ASCII order. 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, commands, diff, sort, uniq

commands — Overview

The following lists the commands included with COHERENT. The command name is given on the left and a description on the right.

CD-ROM Commands

The following commands let you manipulate a CD-ROM device.

cdplayer . . . . . Play audio CDs
cdv . . . . . Interface to CD-ROM devices
cdview . . . . . Read a file from a CD-ROM

Communications

The following commands let you exchange information with other users and other systems.

ckermit . . . . . Interactive inter-system communication and file transfer
cu . . . . . UNIX-compatible interactive communications program
mail . . . . . Send/read electronic mail
mesg . . . . . Permit/deny messages from other users
msg . . . . . Send a brief message to other users
msgs . . . . . Read messages intended for all COHERENT users
uucico . . . . . Connect to a remote system
uucp . . . . . Copy a file to or from a remote system
wall . . . . . Send a message to all logged in users
write . . . . . Converse with another user

De-fragmentation Commands

The following commands give you information about the degree of fragmentation shown by a file system's free list. They can also rebuild a file system, to de-fragment it and so greatly the speed with which you can read and write it.

dpac . . . . . De-fragment a COHERENT file system
fmap . . . . . Measure fragmentation of the free list
qpac . . . . . Map the file system
spac . . . . . Sort a file system
upac . . . . . De-fragment a file system without sorting

Directory and File Handling

The following commands let you create, remove, and otherwise manipulate files and directories.

cat . . . . . Concatenate a file to the standard output
cd . . . . . Change directory
chgrp . . . . . Change the group owner of a file
chmod . . . . . Change the modes of a file
chmog . . . . . Change mode, ownership, and group of a file
chown . . . . . Change ownership of a file
cmp . . . . . Compare bytes of two files
compress . . . . . Compress a file
cp . . . . . Copy a file
cpdir . . . . . Copy directory hierarchy
dd . . . . . Convert the contents of a file
dos . . . . . Manipulate files on MS-DOS file systems
doscat . . . . . Concatenate a file on an MS-DOS file system
doscp . . . . . Copy files to/from an MS-DOS file system
doscpdir . . . . . Copy directories to/from an MS-DOS file system
dosdir . . . . . List the contents of an MS-DOS directory
dosdel . . . . . Delete a file from an MS-DOS file system
dosformat . . . . . Build an MS-DOS file system on a floppy disk
doslabel . . . . . Label an MS-DOS floppy disk
dosls . . . . . List files on an MS-DOS file system
dosmkdir . . . . . Create a directory in an MS-DOS file system
dosrm . . . . . Remove a file from an MS-DOS file system
dosrmdir . . . . . Remove a directory from an MS-DOS file system
fdisk . . . . . View/change hard-disk partitioning
file . . . . . Name a file's type
find . . . . . Search for files satisfying a pattern
gzip . . . . . GNU utility to compress files
gunzip . . . . . GNU utility to uncompress files
l . . . . . List directory's contents in long format
lc . . . . . List directory's contents in columnar format
lf . . . . . List directory's contents in columnar format
ln . . . . . Create a link to a file
lr . . . . . List subdirectorys' contents in columnar format
ls . . . . . List directory's contents
lx . . . . . List directory's contents in columnar format

mkdir . . . . . Create a directory
mv . . . . . Rename files or directories
mvdir . . . . . Rename a directory
pwd . . . . . Print the name of the current directory
qfind . . . . . Quickly find all files with a given name
rm . . . . . Remove files
rmdir . . . . . Remove directories
touch . . . . . Update modification time of a file
uncompress . . . . . Uncompress a file
unpack . . . . . GNU utility to uncompress files
unzip . . . . . Unzip a zipped archive
whereis . . . . . Locate source, binary, and manual files
which . . . . . Locate executable files
zcat . . . . . Concatenate a compressed file
zcmp . . . . . Compare compressed files
zforce . . . . . Force the suffix **.gz** onto every **gzip** file
znew . . . . . Recompress **.Z** files to **.gz** files

Editors

COHERENT includes a number of text editors, to suit a variety of tastes.

ed . . . . . Interactive line editor
elvis . . . . . Berkeley-style screen editor
emacs . . . . . COHERENT screen editor
ex . . . . . Berkeley-style line editor
me . . . . . COHERENT screen editor
sed . . . . . Stream editor
vi . . . . . Berkeley-style screen editor

Games

The following commands are just for fun.

almanac . . . . . Print an almanac entry for this date
banner . . . . . Print large sized letters
cal . . . . . Print a calendar
chase . . . . . Highly amusing video game
fortune . . . . . Print randomly selected, hopefully humorous, text
guess . . . . . Extraordinarily amusing guessing game
lines . . . . . Highly amusing board game
moo . . . . . Greatly amusing numeric guessing game
rubik . . . . . Play Rubik's cube
ttt . . . . . Three-dimensional tic-tac-toe

Kernel Tools

The following commands let you configure the COHERENT kernel, and build a new bootable kernel:

asypatch . . . . . Patch a kernel file for an asynchronous configuration
cohtune . . . . . Set a variable within a device driver
idbld . . . . . Reconfigure the COHERENT kernel
idenable . . . . . Enable or disable a device driver
idmkcoh . . . . . Build a new kernel
idtune . . . . . Set a tunable system value
patch . . . . . Patch a variable or flag within the kernel

Languages and Programming Tools

The COHERENT system comes with a number of languages, and tools for debugging and maintaining your programs.

as . . . . . Mark Williams assembler
asfix . . . . . Convert file to 80386 **as** form
awk . . . . . Report generation, pattern scanning, and processing language
cc . . . . . C-language compiler
cdmp . . . . . Dump COFF files into a readable form

LEXICON

conv . . . . . Numeric base converter
cpp . . . . . C preprocessor
db . . . . . Assembly-level symbolic debugger
ld . . . . . Link relocatable object files
lex. . . . . Lexical analyzer generator
m4 . . . . . Macro processor
make . . . . . Program building discipline
makedepend . . . . . Generate list of dependencies for a **makefile**
nm . . . . . Print a program's symbol table
od . . . . . Print an octal dump of a file
prof . . . . . Print execution profile of a C program
ref. . . . . Display a C function header
srcpath. . . . . Find source files
size . . . . . Print size of an object file
strip . . . . . Strip symbol tables from executable file
yacc. . . . . Parser generator

Libraries and Archives

The following commands help you create and read libraries and archives. These can be used as libraries (such as the libraries used when linking a C program), or to back up files.

ar . . . . . The object librarian/archiver
cpio . . . . . Archiving/backup utility
dump . . . . . File-system backup utility
dumpdate . . . . . Print dump dates
dumpdir . . . . . Print the directory of a dump
gnucpio . . . . . Archiving/backup utility
gtar . . . . . Archiving/backup utility
ranlib . . . . . Create index for object library
restor. . . . . Restore file system
tar. . . . . Archiving/backup utility

Mail

COHERENT comes with with a full-featured, UNIX-style mail facility based on the program **smail**. This is described in the overview article **mail**. The following commands perform mail-related work. Some are also listed in other sections of this article. Please note that the descriptions of **smail** and **rmail** are only for those users who wish to manipulate UUCP mailing on a low level; for most users, the descriptions under the command **mail** are more than sufficient.

checkerr . . . . . Check the mail system for errors
cvmail . . . . . Convert stored mail to System V format
getmap . . . . . De-archive Usenet map articles
lmail . . . . . Deliver local mail
mail. . . . . Send/read electronic mail
mailq . . . . . Display information about spooled mail
mkdbm. . . . . Build a data base for **smail**
mkfnames . . . . . Generate data base of user names
mkhpath. . . . . Build a **pathalias** data base from a **hosts** table
mkline . . . . . Fold mail data into one-line records
mkpath. . . . . Create a pathalias output file
mksort . . . . . Sort the standard input, allowing arbitrarily long lines
newaliases. . . . . Build the **aliases** data base from ASCII source
nptx. . . . . Generate permutations of users' full names
pathalias. . . . . Generate a set of paths among computers"
pathmerge. . . . . Merge sorted paths files
rmail . . . . . Receive mail
rsmtp. . . . . Run batched SMTP mail
runq. . . . . Periodically process the mail queue
savelog . . . . . Save a mail log
smail . . . . . Send mail
smtpd . . . . . SMTP daemon

For information on the configuration files used by the **smail** system, see the overview article **mail**, or the article **Administering COHERENT**.

Printing

The following commands help you print text. For commands that drive communications devices, e.g., modems, see the section on *Communications*, above.

cancel . . . . . Cancel a print job
chreq . . . . . Change priory, lifetime, or printer for a job
epson . . . . . Prepare a file for an Epson printer
fnkey . . . . . Set/print function keys for the console
hp . . . . . Prepare files for HP LaserJet-compatible printer
hpr . . . . . Send to LaserJet printer spooler
hpskip . . . . . Abort/restart current listing on LaserJet
lp . . . . . Spool a job for printing
lpadmin . . . . . Administer the **lp** print-spooler system
lpsched . . . . . Print jobs spooled with command **lp**
lpshut . . . . . Turn off the printer daemon
lpr . . . . . Send to line printer spooler
lpskip . . . . . Terminate/restart current line printer listing
lpstat . . . . . Give status of printer or job
pclfont . . . . . Prepare a PCL font for downloading via MLP
reprint . . . . . Reprint a spooled print job
route . . . . . Show or reset a user's default printer
stty . . . . . Set/print terminal modes

Shell Commands

COHERENT comes with two command interpreters, or *shells*: **ksh**, the Korn shell, and **sh**, the Bourne shell. The following commands are used either by the Korn shell, by the Bourne shell, or by both. Please note that commands used only by the Korn shell are marked by a dagger '†', whereas commands used only by the Bourne shell are marked by an asterisk '*'.

alias† . . . . . Set an alias
basename . . . . . Strip path information from a file name
bind† . . . . . Bind key sequence to editing command
break . . . . . Exit from shell construct
builtin† . . . . . Execute a command as a built-in command
case . . . . . Execute commands conditionally according to pattern
cd . . . . . Change directory
continue . . . . . Terminate current iteration of shell construct
dirname . . . . . Extract a directory name
dirs* . . . . . Print contents of directory stack
echo . . . . . Repeat an argument
eval . . . . . Evaluate arguments
exec . . . . . Execute command directly
exit . . . . . Exit from a shell
export . . . . . Add a shell variable to the environment
expr . . . . . Compute a command line expression
false . . . . . Unconditional failure
fc† . . . . . Edit and re-execute one or more previous commands
for . . . . . Execute commands for tokens in list
from . . . . . Generate list of numbers, for use in loop
getopts . . . . . Parse command-line options
hash† . . . . . Add a command to the shell's hash table
id . . . . . Print user and group IDs and names
if . . . . . Execute a command conditionally
jobs† . . . . . Print information about jobs
let . . . . . Evaluate an expression
nohup . . . . . Run a command while ignoring hangup signals
popd* . . . . . Pop an item from the directory stack
prep . . . . . Produce a word list
print† . . . . . Echo text onto the standard output

LEXICON

pushd* . . . . . Push an item onto the directory stack
read . . . . . Assign values to shell variables
readonly . . . . . Mark a shell variable as read only
set . . . . . Set shell option flags and positional parameters
shift . . . . . Shift positional parameters
sleep . . . . . Stop executing for a specified time
tee . . . . . Copy input to multiple output streams
test . . . . . Evaluate conditional expression
times . . . . . Print total user and system times
trap . . . . . Execute command on receipt of signal
true . . . . . Unconditional success
typeset† . . . . . Set/list variables and their attributes
umask . . . . . Set the file-creation mask
unalias† . . . . . Remove an alias
unset . . . . . Unset an environmental variable or shell function
until . . . . . Execute commands repeatedly
wait . . . . . Await completion of background process
whence† . . . . . List a command's type
while . . . . . Execute commands repeatedly
xargs . . . . . Execute a command with many arguments

String Processing

Some of the most useful commands are those that process strings. COHERENT has many commands that search for strings, manipulate strings, sort strings, and otherwise perform useful manipulations on strings.

c . . . . . Print multi-column output
cgrep . . . . . Pattern search for C programs
comm . . . . . Print common lines
cut . . . . . Select portions of each line of a file
detab . . . . . Replace tab characters with spaces
diff . . . . . Summarize differences between two files
diff3 . . . . . Summarize differences among three files
egrep . . . . . Extended pattern search
grep . . . . . Pattern search
head . . . . . Print the beginning of a file
join . . . . . Join two data bases
look . . . . . Find matching lines in a sorted file
more . . . . . Display text one screenful at a time
paste . . . . . Merge lines of files
rev . . . . . Print text backwards
scat . . . . . Print text files one screenful at a time
sort . . . . . Sort lines of text
split . . . . . Split a text file into smaller files
strings . . . . . Print all character strings from a file
tail . . . . . Print the end of a file
tr . . . . . Translate characters
tsort . . . . . Topological sort
uniq . . . . . Remove/count repeated lines in a sorted file
view . . . . . Berkeley-style text viewer
wc . . . . . Count words, lines, and characters in text files
zdiff . . . . . Compare two compressed files
zgrep . . . . . Search compressed files for a regular expression
zmore . . . . . Display compressed text one page at a time

System Accounting

The following commands help you to keep track of how your COHERENT system is working.

ac . . . . . Summarize login accounting information
accton . . . . . Enable/disable process accounting
df . . . . . Measure free space on disk
du . . . . . Summarize disk usage
hmon . . . . . Monitor the COHERENT System

ps . . . . . Print process status
sa . . . . . Print a summary of process accounting
quot. . . . . Summarize file-system usage
time. . . . . Time the execution of a command
times . . . . . Print total user and system times
uulog . . . . . Examine UUCP operations

System Maintenance

These commands help you to maintain your COHERENT system.

asymkdev . . . . . Create nodes for asynchronous devices
at . . . . . Execute commands at given time
bad . . . . . Maintain list of bad blocks
badscan . . . . . Examine a device for bad blocks
build . . . . . Install COHERENT onto a hard disk
check. . . . . Check file system
clri . . . . . Clear i-node
crontab. . . . . Copy a command file into the crontab directory
date. . . . . Print/set the date and time
dcheck . . . . . Check directory consistency
fdformat . . . . . Low-level format a floppy disk
fsck . . . . . Check and repair file systems interactively
icheck . . . . . i-node consistency check
mkfs . . . . . Make a new file system
mknod . . . . . Make a special file or named pipe
mount . . . . . Mount a file system
ncheck . . . . . Print file names corresponding to i-node
newgrp . . . . . Change to a new group
newusr . . . . . Add new user to COHERENT system
reboot . . . . . Reboot the COHERENT system
shutdown . . . . . Shut down the COHERENT system
sync. . . . . Flush system buffers
ttytype . . . . . Set default terminal types
umount . . . . . Unmount a file system
uuchk . . . . . Sanity-check the UUCP system

terminfo

COHERENT supports an implementation of **terminfo**, the terminal-description utility used under UNIX System V. (It also supports **termcap**, should you prefer to use that venerable, but still useful, system.) The following commands help support **terminfo**:

captoinfo . . . . . Convert termcap data to **terminfo** form
infocmp . . . . . De-compile a **terminfo** binary file
tic . . . . . Compile a **terminfo** description

Text Processors

These commands help you to create orderly, attractive printed text. For information on how to print the output of these commands, see the commands listed under *Device Handling*, above.

col. . . . . Remove reverse and half line motions
deroff. . . . . Remove text formatting control information
nroff . . . . . Text-formatting language
fmt . . . . . Adjust the length of lines in a file of text
fwtable . . . . . Build a font-width table from PCL or PostScript font
lcasep. . . . . Convert text to lower case
pr . . . . . Paginate and print files
prps. . . . . Paginate and print files on PostScript printers
PSfont . . . . . Cook an Adobe font into PostScript format
spell. . . . . Find spelling errors
troff. . . . . Extended text-formatting language
typo. . . . . Detect possible typographical and spelling errors

UUCP

The UUCP commands lets you form a network with other COHERENT or UNIX systems. Members of the network can grant each other permission to exchange mail and execute commands on each others' systems remotely and automatically, without having to be directed by a human being. The overview article **UUCP** describes the COHERENT UUCP facility in some detail. The following commands perform UUCP-related work; note that some of the commands listed here also are also listed in other sections of this article.

mwcbbs . . . . . Download files from the Mark Williams bulletin board
uuchk . . . . . Sanity-check the UUCP system
uucico . . . . . Connect to a remote system
uuconv . . . . . Convert UUCP configuration files into Taylor format
uucp . . . . . Copy a file to or from a remote system
uudecode . . . . . Decode a transmitted UUCP file
uuencode . . . . . Encode a UUCP file for tranmission
uuinstall . . . . . Configure UUCP control files
uumkdir . . . . . Create UUCP directories
uulog . . . . . Examine UUCP operations
uumvlog . . . . . Archive UUCP log files
uuname . . . . . Print names of recognized systems
uupick . . . . . Pick up a file uploaded from a remote system
uurmlock . . . . . Remove UUCP lock files
uusched . . . . . Call all systems that have jobs waiting for them
uuto . . . . . Send a file to a remote system
uutouch . . . . . Force polling of a remote site
uux . . . . . Execute a command on a remote system
uuxqt . . . . . Execute file as requested by remote system

Miscellaneous

The following commands do not fit neatly into any of the above categories. These include some of the more interesting and useful COHERENT commands, and are worth your attention.

apropos . . . . . Find manual pages on a given topic
ATclock . . . . . Read/set the AT realtime clock
bc . . . . . Interactive calculator with arbitrary precision
calendar . . . . . Electronic reminder service
chroot . . . . . Change root directory
clear . . . . . Clear your terminal's screen
coh_intro . . . . . Tour the COHERENT file system
crypt . . . . . Encrypt/decrypt text
dc . . . . . Desk calculator
disable . . . . . Disable a port
elvprsv . . . . . Preserve the modified version of a file after a crash
elvrec . . . . . Recover the modified version of a file after a crash
enable . . . . . Enable a port
env . . . . . Execute a command in an environment
factor . . . . . Factor a number
findmouse . . . . . Examine a port to see if a mouse is plugged into it
ftbad . . . . . Manipulate bad-block list on a floppy-tape cartridge
help . . . . . Print concise description of command
ideinfo . . . . . Display information about an IDE disk drive
install . . . . . Install a software update onto COHERENT
ipcrm . . . . . Remove an interprocess-communication memory item
ipcs . . . . . Display a snapshot of interprocess communications
kill . . . . . Signal a process
ksh . . . . . Invoke the Korn shell
login . . . . . Log in or change user name
makeboot . . . . . Make a bootable floppy disk
man . . . . . Display Lexicon entries
mklost+found . . . . . Make an enlarged **lost+found** directory
passwd . . . . . Set/change login password
phone . . . . . Print numbers and addresses from phone directory
script . . . . . Capture a terminal session into a file

sh . . . . . Invoke the Bourne shell
su . . . . . Substitute user id, become superuser
sum . . . . . Print checksum of a file
tape . . . . . Manipulate a tape device
tty . . . . . Print the user's terminal name
ttystat . . . . . Get terminal status
uname . . . . . Print information about the system
units . . . . . Convert units of measure
vsh . . . . . Invoke the COHERENT visual shell
who . . . . . Print who is logged in
yes . . . . . Print infinitely many responses

For more information on any of these commands, see its entry within the Lexicon.

See Also

Administering COHERENT, Programming COHERENT, Using COHERENT

compress — Command

Compress a file
compress [-dfvc] [-b*num* **] [** *file ...* **]**

compress compresses a file using the Lempel-Ziv algorithm. With text files and archives, it often can achieve 50% rate of compression.

If one or more *file*s are specified on the command, **compress** compresses them and appends the suffix **.Z** onto the end of each compressed file's name. If no *file* is specified on the command line, **compress** compresses text from the standard input and writes the compressed text to the standard output.

compress recognizes the following options:

-b The "bits" option. **compress** uses the compression level set via the *num* argument. Previous releases of **compress** would only allow values of *num* up to 12, with 12 being the default value if the **-b** option was not specified. The version of **compress** introduced with COHERENT version 3.1 handles values up to 16, with 12 being the default.

-c Send output to stdout.

-d Decompress rather than compress.

-f Force an output file to be generated even if no space is saved by compression.

-v Verbose mode: force **compress** to write statistics about its performance.

If you wish to ensure backwards compatibility with previous releases of COHERENT, do not use **compress** with a *num* value greater than 12.

See Also

commands, compression, gzip, ram, uncompress, zcat

compression — Technical Information

Programs used to compress text

Compression is the technique whereby a file is analyzed mathematically and made smaller. Compress is very useful in reducing the amount of disk space taken up by large files that you use infrequently.

The amount of compression will vary, depending upon the type of file being compressed, the compression algorithm used, and the level of compression requested. In general, files that show a great deal of repetition internally will compress more thoroughly than those that are largely random; thus, in general a text file will compress more thoroughly than will a digitized sound sample or image (although there are exceptions). The higher the level of compression you request, the more thoroughly the file will be compressed, but the longer the machine will have to work to achieve it. In most instances, raising the level of compression very high will save only a few bytes at a great cost in computer time.

You should note, too, that although compression algorithms try very hard not to lose information, it is possible that compressing some very complex files may result in a loss of information: that is, if you compress a file and de-compress it, the de-compressed file may be exactly the same as it was before you first compressed it. These programs will not affect most everyday varieties of data; but you should be aware of this fact.

COHERENT comes with the following tools for compressing and uncompressing files:

compress
This program compresses files uses the Lempel-Ziv algorithm. By default, it creates a file with the suffix **.Z**. It replaces the uncompressed original with its compressed analogue.

gtar This program creates tape archives. Its options **-z** and **-Z** invoke, respectively, the programs **gzip** and **compress** to compress the archive as it is being built, thus permitting you to build a compressed archive automatically.

gunzip This de-compresses files that had been compressed by the program **gzip**. By default, it works only with files that have the suffices **.z** or **.gz**. It replaces the compressed file with its uncompressed analogue.

gzip This program compresses files into the **zip** format. In general, it is faster and more thorough than **compress**, although it may not work as well on some files. It replaces the uncompressed original file with its compressed analogue.

uncompress
This uncompresses files that had been compressed by **compress**. It works with files that have the suffix **.Z**. It replaces the compressed file with its uncompressed analogue.

zcat This program de-compresses "on the fly" programs that had been compressed by **compress**, and writes the decompressed form to the standard output device. This is useful if you want to look at the contents of a compressed file but do not want to bother with de-compressing all of it.

Default Suffixes

Compressed files cannot be used in their compressed form; you must first uncompress them before you can use them. The key to uncompressing a compressed file is figuring out what program it was compressed with in the first place, so you can apply the correct de-compression tool.

If you have received a compressed file from a third-party source, you may have no idea what tool was used to compress the file; fortunately, however, most compression tools use standard suffices to "stamp" the files they compress. The following table gives commonly used suffices, plus examples of how to uncompress files that bear them:

Suffix	*Compression Program*	*Decompression Program*	*Example*
.Z	**compress**	**uncompress**	**uncompress foo.Z**
.tar.Z	**tar|compress**	**uncompress|tar**	**zcat foo.tar.Z | tar xvf -**
.z	**gzip**	**gunzip**	**gunzip foo.z**
.tar.z	**tar|gzip**	**gunzip|tar**	**gunzip foo.tar.z ; tar xvf foo**
.tgz	**gtar -cz**	**gtar -xz**	**gtar -xvzf foo.z**
.gtz	Same as **.tgz**		
.taz	Same as **.tgz**		

See Also

compress, gtar, gunzip, gzip, uncompress, Using COHERENT, zcat

con.h — Header File

Configure device drivers
#include <sys/con.h>

The header file **con.h** gives the configuration for each device driver included with the COHERENT system. Each driver is defined using the structure **CON**, which is declared in **<sys/con.h>**.

See Also

header files

config — System Administration

File that configures smail
/usr/lib/mail/config

File **/usr/lib/mail/config** holds instructions that configure the mailer-delivery program **smail**. You can modify this file to supplement, modify, or override **smail**'s default configuration.

Please note that this file is in no way related to file **/usr/lib/uucp/config**, which can be used to configure the Taylor UUCP system. For details on how to configure UUCP, see the Lexicon entry for **/usr/lib/mail/config**, which immediately follows this article in the Lexicon.

The rest of this article describes **config**, the attributes you can set within it, and how the setting of each attribute affects **smail**'s behavior.

Suite of Configuration Files

To begin, your machine can have two **smail** configuration files: a primary one, and a secondary one. Either can reset the values of any **smail** variable; for example, each can define names for the local host, define where files reside, or set the values for site-definable message-header fields. You are not obliged to use a configuration file: if **smail**'s default configuration suits you, then you can rename or move the primary configuration file so it will no longer be read. Likewise, if you have a primary configuration file, you are not obliged to have a secondary one.

smail reads the primary configuration file first, then the secondary configuration file. The values in the secondary configuration file can override those set in the primary file; the primary file, in turn, can redefine the name of the secondary configuration file. This gives you great flexibility to configure **smail** to suit your needs and preferences.

Format of a Configuration File

A configuration file consists of instructions; each instruction, in turn, sets an *attribute* to a value. Attributes come in three flavors: *string*, *numeric*, and *Boolean*. To set a variable to a string or numeric value, use the form:

variable **=** *value*

For example, the instructions

```
postmaster = tron@glotz.uucp
domains = wall.com
spool_mode = 0664
```

set the default address for the postmaster to **tron@glotz.uucp**, the attribute **domains** to **wall.com**, and the permissions for spool files to permit the file's owner and group to write into it.

Boolean attributes are either turned off or turned on. To turn on a Boolean attribute, use the notation:

+*boolean-attribute*

To turn it off, use the notation:

-*boolean-attribute*

You can also use the notation *-attribute* to set a numeric variable to zero and to un-set a value for a string variable. For example, the following instructions disable the use of an external transport file and tells **smail** that configuration files are not optional:

```
-transport_file
+require_configs
```

smail ignores blanks lines within a configuration file. If **smail** encounters a '#' character, it ignores that character plus all text to its right; thus, you can use this character to introduce a comment.

If a line begins with white space, **smail** assumes that it continues the previous line; in this way, you can extend an instruction over more than one line. For example, the following instructions set the **Received:** header field to use for messages to a multi-line value, and also set the name of a user that has few access capabilities:

```
# Use a verbose format for the Received: header field
received_field = "Received: by $primary_name
      with smail ($version_string)
      id <$message_id@$primary_name); $date"

nobody = unknown # comment: user "unknown" has few access capabilities
```

smail Attributes

The following names the attributes that you can set in a configuration file. Each attribute's name is followed by its type and its default setting in parentheses.

auth_domains (string, off)
: Name the domains for which your host is considered authoritative — i.e., the domains that your host knows how to access directly. The domain names must be separated by a single colon character ':'. Mail

addressed to any domain named in this list will not be forwarded to the smart host (described below).

auto_mkdir (Boolean, on)
If set, **smail** creates all directories required for spooling and logging if they do not exist. However, **smail** will never create required parent directories.

auto_mkdir_mode (integer, 0755)
When **smail** creates a directory, give it this permission mask. For details on what the numbers in a permission mask mean, see the Lexicon entry for **chmod**.

console (string, **/dev/console**)
Name the console device. This device is used as a last resort in attempting to write panic messages.

copying_file (string, **COPYING**)
The path name to file **COPYING**, which states your distribution rights and details the warranty information from the authors of **smail**. If this does not begin with '/', **smail** assumes that it is in the directory named by attribute **smail_lib_dir** (described below).

date_field (string, **Date: $spool_date**)
smail expands this string to form field **Date:** in a mail message's header, should the header not already contain such a field.

delivery_mode (string, **foreground**)
The default mode for delivering new mail. This can be one of the following values:

foreground
Immediate delivery via the process that received the message.
background
Immediate delivery via a child process. The process that received the message exits immediately.
queued
Do not attempt delivery until a later queue run.

director_file (string, directors)
This names the file that configures **smail**'s directors. If this does not begin with '/', **smail** assumes that it is in the directory named by variable **smail_lib_dir** (described below).

domains (string)
This sets the domain name that **smail** writes into the header of an outgoing mail message. It is computed at run time. If attribute **visible_name** is turned off, then **smail** sets it to the first name set by attribute **hostnames**. If **hostnames** is not set, then **smail** constructs the domain-name host names of the form *hostname.domain*. *hostname* is set in file **/etc/uucpname** *domain* is a name set by the attribute **domains**— **smail** uses each entry in **domains**, in order, to create the *hostnames* value.

For sites in the **UUCP** zone, **domains** often will merely be set to the string **uucp**. Finally, you can use the variable **$visible_name** within the string to which you set this attribute.

For compatibility with earlier versions of **smail**, this attribute can also be called **visible_domains**.

error_copy_postmaster (Boolean, off)
Send the postmaster a copy of every error message. Normally, **smail** sends the postmaster only the errors that appear to result from administrator mistakes. If you set this attribute, then **smail** also sends the postmaster the errors that are returned to the sender or that are mailed to owners of mailing lists.

fnlock_interval (number, 3)
Set the sleep interval between retries while attempting to lock mailbox files with a lockfile-based locking protocol. Under COHERENT, the function **sleep()** has a one-second granularity; therefore, you must this value to at least two.

fnlock_mode (number, 0666)
Create mailbox lock files.

fnlock_retries (number, five)
The number of times **smail** attempts to lock mailbox files using a file-based locking protocol.

from_field (string)
smail expands this string to form the fields **From:** and **Sender:** in a mail message's header. The expanded string must begin with **From:**, which may be replaced by other strings to form an actual header field. The default value is:

```
From: $sender${if def:sender_name: ($sender_name)}
```

grades (string)
Set the grade (or priority) characters that correspond to values of the **Precedence:** field in a mail message's header. The fields within the string are separated by ':'; precedence strings alternate with grade characters. Numbers have higher priority than upper-case letters, which in turn are higher than lower-case letters. Lower numbers are higher in priority than higher numbers, and the same goes for letters lower in the alphabet. Grades in the range 'a' through 'm' only return an error message and header to the sender when an error occurs. Grades in the range 'n' through 'z' return nothing to the sender should an error occur. The precedence names recognized by many BSD **sendmail** configurations are **special-delivery**, **first-class**, and **junk**. Others are useful mainly for getting mail out of the local machine or for communicating with other machines that run **smail** in a similar configuration. The grade character for a message is available in string expansions as the variable **$grade**. The default setting is:

```
special-delivery:9:air-mail:A:first-class:C:bulk:a:junk:n
```

hit_table_len (number, 241)
The length of the internal-address "hit" table. **smail** hashes addresses into this table to prevent multiple deliveries to the same address. Longer tables speed address hashing, at the price of a small increase in the amount of memory used. NB, this value may be ignored in the future.

host_lock_timeout (numeric, 30)
Set the time during which **smail** will attempt to lock a host's retry file; this file is used to guarantee exclusive delivery to that thost. If **smail** cannot lock the file within this time, then it leaves the message in the queue, to be delivered later.

A number with no suffix indicates seconds. Suffixes can be added to indicate a time multiplier: **m** indicates minutes, **h** indicates hours, and **d** indicates days.

hostnames (string)
hostname (string)
A colon-separated list of names for the local host. This list, together with the attributes **uucp_host** and **more_hostnames**, should represent all possible names for the local host. Note that **smail** does not recognize the name **hostname** as a name for the local host unless that name is also set by one of the other **hostname** variables. If your local host is in more than one domain or can gateway to more than one level of domains, then this attribute should represent those names. For a host in a registered domain in the UUCP zone, which is also in the maps distributed over USENET, **localhost.uucp** should also be in the list. The first value in **hostnames** is used internally as a special "primary name" for the local host.

Under COHERENT, this attribute is turned off by default. **smail** computes the value of **hostnames** by pairing the local host's name, as set in file **/etc/uucpname**, with every value set by attribute **domains**. **smail** re-computes the default value each time you run it.

lock_by_name (Boolean, on)
If this variable is turned on, locking of the input spool file is always based on lock files. Otherwise, an i-node—based locking mechanism may be used, such as the BSD function **flock()** or **lockf()** under System V or COHERENT. I-node—based locking is more efficient, if available. However, lock files can be easily created by shell scripts, which may be advantageous under some circumstances.

lock_mode (number, 0444)

log_mode (number, 0664)
The mode assigned to newly created mail-system log files.

logfile (string, **/usr/spool/smail/log/logfile**)
The file into which **smail** writes transaction messages and error messages. If this file does not exist, **smail** creates it with the mode set by variable **log_mode**.

max_hop_count (number, 20)
If the hop count for a message equals or exceeds this number, then any attempt at remote delivery results in an error message being returned to the sender. **smail** uses this mechanism to prevent infinite loops. To set the hop count for a specific message, use **smail**'s command-line option **-h**. Otherwise, **smail** computes it from the number of **Received:** fields in the message header.

max_load_ave (number)
For systems on which a load average can be computed, this attribute sets the maximum load average at which mail will be delivered. If the load average exceeds this number, **smail** saves incoming mail within the input spool directory for delivery later. Under COHERENT, this attribute is not set; therefore, **smail** does not compute the load average, and always attempts to deliver mail.

max_message_size (number, **100k**)
Set the maximum size of a message. **smail** truncates messages longer than this. (This is not yet implemented; at present, **smail** sets nolimit on the size of a message.)

message_buf_size (number, **100k**)
The size of the internal buffer that **smail** uses to read and write messages. The larger the value of this buffer, the fewer the number of calls to **read()** are required to read the message, because the entire message is always kept in memory. The default value is 100 kilobytes (**100k**).

message_id_field (string)
smail expands this attribute to form the field **Message-Id:** in a mail message's header. This will be used if such a field does not already exist in the header. The default value is:

```
Message-Id: <$message_id@$primary_name>
```

message_log_mode (number, 0644)
Each message has associated with it a unique file that contains a transaction log for that message. This number sets the permissions that **smail** gives this file when it creates it.

method_dir (string, **methods**)
If a method attribute for a router does not specify a path name that begins with '/', **smail** prefixes this directory onto the path to form the complete path for the method file. If this does not begin with '/', **smail** assumes that it is in the directory set by attribute **smail_lib_dir** (described below). See the description of the router file for more information on method files.

more_hostnames (string, off)
A colon-separated list of host names. These host names are in addition to any names that **smail** computed from the domains when forming the value of the variable **hostnames**. Thus, it is useful for specifying names that are not formed from the computed name for the host.

Attribute **more_hostnames** can also be called **gateway_names**, because it is often used to indicate the list of domains for which this machine is a gateway.

nobody (string, **nobody**)
The default user. This variables defines permissions to use when no other user is specified. Also, **smail** uses this user in some conditions when it is not certain whether a set of actions can be trusted, if performed under other, potentially more powerful users. This should reference a login identifier that has little power to do harm or access protected files.

paniclog (string, **/usr/spool/smail/log/paniclog**)
The name of the file onto which **smail** appends panic messages and other important error messages. If this file does not exist, **smail** creates it and assigns it the permissions specified by variable **log_mode**. **smail** records in this log all errors that require human intervention, such as configuration errors or directory-permission errors, that prevent mail spooling or delivery.

When a configuration error occurs, **smail** usually moves the mail into a special error directory under the input spool directory. This prevents **smail** from again attempting to delivery the message until the configuration error has been corrected.

Thus, you should regularly check both the panic log and the error directory, especially after you have changed a configuration. When the problem has been resolved, you can move the diverted messages back into the spool directory, and **smail** will again attempt to deliver them.

postmaster_address (string, **root**)
postmaster (string, **root**)
This attribute sets the default address of the postmaster. If the address **Postmaster** is not resolved by any of the configured directors, **smail** then uses this address.

qualify_file (string, **qualify**)
This variable names the file that contains the host-name qualification information. If this does not begin with '/', **smail** assumes that it is a subdirectory of the directory defined by the attribute **smail_lib_dir**.

queue_only (Boolean, off)
If this flag is set, then **smail** does not deliver incoming mail immediately. It only attempts delivery when it explicitly processes the input queue, such as when you invoke it with command-line option **-q**.

received_field (string)
smail expands this string to form the field **Received:** in a mail message's header. It inserts this field into the header if the "received" attribute is not explicitly turned off for a transport. The default value for **received_field** is:

```
received_field="Received: \
      ${if def:sender_host\
            {from $sender_host by $primary_name\
             ${if def:sender_proto: with $sender_proto}\
             \n\t(Smail$version #$compile_num) }\
      else {by $primary_name ${if def:sender_proto:with $sender_proto }\
             (Smail$version #$compile_num)\n\t}}\
      id $message_id; $spool_date"
```

require_configs (Boolean, off)
If this option is turned off or is not set, then **smail** does not require its configuration files to exist. This applies to the primary and secondary configuration files, and the director, router, and transport files (respectively, **/usr/lib/mail/directors**, **/usr/lib/mail/routers**, and **/usr/lib/mail/transports**). If one of these files does not exist, **smail** ignores it and instead uses its internally compiled configuration. If, however, you turn on this attribute, then if **smail** cannot find a configuration file whose file name is not null, it displays a panic message and exits.

To set a configuration file's name to null, turn off the attribute that names it. For example, to set the router file's name to null, use the attribute **-router**.

retry_file (string, **retry**)
This names the file that contains the retry-control information. If this name does not begin with '/', **smail** assumes that it is in directory named by variable **smail_lib_dir** (described below).

retry_duration (interval, **5d**)
This specifies the default period of time for which **smail** will attempt to deliver a message. If the message cannot be delivered within this period of time, **smail** assumes it is undeliverable, and sends a "bounce" message either to the sender or to the list's owner, should there be one. A number with no suffix indicates seconds. Suffices can be added to indicate a time multiplier: **m** indicates minutes, **h** indicates hours, and **d** indicates days. Under COHERENT, the default is five days.

retry_interval (interval, **10m**)
If **smail** cannot connect to a given host, it will wait at least this amount of time before it tries again. This applies to all messages routed to the host in question, to help process a queue efficiently.

return_path_field (string, **Return-Path: <$sender>**)
smail expands this string into field **Return-Path:** in the mail-message's header. It inserts this field into the header if attribute **return_path** is turned on for a given transport in file **/usr/lib/mail/transports**.

router_file (string, **routers**)
This attribute names the file that contains the router-configuration information. If this does not begin with '/', **smail** assumes that it is in the directory named by attribute **smail_lib_dir** (described below).

second_config_file (string, none)
This names the secondary configuration file. The section on configuration files, above, describes how this file relates to the primary configuration file. If this file's name does not begin with '/', **smail** assumes that it is in the directory named by attribute **smail_lib_dir** (described below).

This is primarily useful in networks whose machines share file systems. In particular, the attributes **smart_user**, **smart_path**, and **smart_transport** are set in the secondary configuration file.

sender_env_variable (string, not set)
This attribute names the environmental variable that, in turn, gives the name of the mail message's sender. Normally, the name of the sender is determined from her login identifier, or by checking calling process's real-user identifier. If **sender_env_variable** is set and the environmental variable it names exists, then **smail** uses that name by default. For example, if the line

```
sender_env_variable=BOGUS_NAME
```

appears in **/usr/lib/mail/config**, and if variable **BOGUS_NAME** is set in the user's environment, then **smail** uses that name to identify the sender, instead of the name for that user that appears in file **/etc/passwd**.

smail (string, **/bin/smail**)
This attribute names the **smail** binary. **smail** uses this to re-**exec** itself when a major configuration change has been detected, or to **exec smail** when delivering error messages. If this name does not begin with '/', **smail** assumes that this binary is kept in the directory named by attribute **smail_lib_dir**.

smail_lib_dir (string, **/usr/lib/mail**)
This attribute gives the full path name of the directory in which **smail** by default seeks its configuration files.

smail_util_dir (string, **/usr/lib/mail**)
This attribute gives the full path name of the directory that holds **smail**'s utilities, in particular the utilities **mkaliases** and **mkdbm**.

smart_path (string, not set)
This attribute defines the value that the router **smarthost** uses by default for its **path** attribute. It gives the path to a machine whose routing data base is more complete than the one on your local host. By default, this is not set; however, if you using UUCP to receive mail service from another system, you must set this variable to the name of that system. For details, see the Lexicon entry for **routers**.

smart_transport (string, not set)
This attribute defines the value that the **smarthost** router driver uses by default for its attribute **transport**. For details, see the Lexicon entry for **routers**.

smart_user (string, not set)
This attribute defines the value that the **smarthost** router driver uses by default for its attribute **smart_user**. For details, see the Lexicon entry for **routers**.

smtp_accept_max (number, 20)
This attribute sets the maximum number of SMTP connections that **smail** will process at any one time. This is for use with SMTP daemons started with **smail**'s command-line option **-bd**, or through the command **smtpd**. If **smail** receives a a connection request when this number of SMTP-connection children have already been forked, **smail** shuts down the connection with SMTP message 421. If this attribute is set to zero, then the number of SMTP connections is unlimited.

smtp_accept_queue (number, 5)
If this number of SMTP connection processes is exceeded, then **smail** accepts additional connections but queues their messages for later processing. When the number of current connection processes drops below this number, **smail** resumes the immediate processing of mail (if attribute **delivery_mode** is set to **foreground** or **background**.) If **delivery_mode** is set to zero, then **smail** will always process mail immediately, regardless of the number SMTP connections that it is handling. Note that the value of **smtp_accept_queue** should be less than the value of **smtp_accept_max**. Setting **smtp_accept_max** to zero prevents **smtp_accept_queue** from working correctly in all cases.

smtp_banner (string)
smail expands this string to the SMTP startup banner. **smail**'s SMTP server writes this banner when it accepts a connection request. Each line of this message is automatically preceded by identification code "220"; newlines are correctly changed into a carriage-return newline sequence. The default value for **smtp_banner** is:

```
$primary_name Smail$version #$compile_num ready at $date
```

smtp_debug (Boolean, on)
This Boolean variable controls the meaning of the **DEBUG** command when receiving SMTP commands. If this variable is on, then the **DEBUG** command (with an optional debugging level) sets debugging to the specified level, or to level 1 if no level was specified. **smail** writes the debugging output to the SMTP connection.

smtp_receive_command_timeout (interval, **5m**)
This attribute sets the time that **smail**'s SMTP daemon waits for a **receiver** command after it displays its prompt. If the daemon does not receive the command within this interval, it closes down the connection

and exits. The default is **5m**, that is, five minutes.

smtp_receive_message_timeout(interval, 2h)
This attribute sets the time that **smail** SMTP daemon waits for a message after it has displayed its prompt:

```
354 Enter mail
```

If it does not receive the entire message within this interval, it removes the message, closes the connection, and exits. The default is **2h**, that is, two hours.

spool_dirs (string, **/usr/spool/smail**)
This sets the directory or directories into which **smail** spools incoming mail. If it names more than one directory, the directories must be separated by a colon ':'. If **smail** cannot write a message to the first directory (say, due to permission problems, file-system-full errors, etc.), it tries to write the message into the other directories, one after another, until it either succeeds in writing the message or runs out of directories to try. Each spool directory is expected to have the following writable subdirectories:

input	The actual spool files
lock	Temporary lock files
msglog	Temporary per-message transaction logs and audit trails
error	Messages failing from problems requiring human intervention

spool_grade (character, **C**)
This attribute gives the default grade for mail messages. It can be overridden by a **Precedence:** field in a message's header. **smail** uses the grade to sort messages in the input spool directory. The grade is also available in string expansions as the variable **$grade**. See the description of the attribute **grades**, above, for more information.

spool_mode (number, **0440**)
This attribute sets the permissions **smail** gives to spooled files.

transport_file (string, **transports**)
This attribute names the file that holds the transport-configuration information. If the directory does not begin with '/', **smail** assumes it is in the directory named by the attribute **smail_lib_dir** (described above).

trusted_users (string, off)
This names the users who are trusted to specify a sender for a message. Users who are not in this list cannot specify a **Sender:** field in a mail header; if they do, **smail** removes it. If a trusted user specifies a **From:** header field, then **smail** also creates a **Sender:** field that names the real user who submitted the message.

In general, this attribute should name every user under whom remote mail is received and sent to **smail**. If this list is turned off, using the form **-trusted**, then every user is trusted.

NB, **smail** uses the real user identifier to verify a trusted user. However, the program **uucico** runs under the real user identifier of the user who invoked it — and any user can invoke **uucico**. **smail** cannot distinguish this case from any other, and thus will do the "wrong thing" in this instance. Under COHERENT, this attribute is turned off by default to avoid this problem.

trusted_groups (string, off)
This attribute names the user groups that are trusted to specify a sender of a message. **smail** checks a user's effective group identifier to ensure that he really is a member of a trusted group. Thus, were **smail** a **setgid** program, then this string would be of no value and should be turned off. However, if **smail** is not set gid (as it is not under COHERENT), then programs that invoke **smail** under a specific effective gid, not a specific real uid, can be detected and can be properly treated as trusted.

uucp_name (string)
This attribute gives the name of your local host. It is computed at run-time. This name is available in string expansions as the variable **$uucp_name smail** also uses it in the "remote from *hostname*" suffix to "From" lines for mail being delivered to remote machines, when the **from** attribute is turned on for a transport.

visible_domains (string)
This is a synonym for attribute **domains**.

See Also

Administering COHERENT, directors, mail [overview], routers, smail, transports

Notes

Copyright © 1987, 1988 Ronald S. Karr and Landon Curt Noll. Copyright © 1992 Ronald S. Karr.

For details on the distribution rights and restrictions associated with this software, see file **COPYING**, which is included with the source code to the **smail** system; or type the command: **smail -bc**.

config — System Administration

File that configures UUCP
/usr/lib/uucp/config

The file **/usr/lib/uucp/config** performs overall configuration of the Taylor UUCP system. By setting commands within this file, you can override the default settings that are compiled into the COHERENT edition of UUCP.

Please note that this file is in no way related to file **/usr/lib/mail/config**, which configures **smail**, the mail-delivery program. For details on how to configure **smail**, see the Lexicon entry for **/usr/lib/mail/config**, which immediately precedes this article in the Lexicon.

Please note also that COHERENT does *not* include an edition of this file with its release of Taylor UUCP. That is because the default behaviors for COHERENT are already compiled into UUCP. However, you can create this file if you wish, and use it to change or override the default behaviors built into Taylor UUCP. This lets you customize UUCP to suit your needs and preferences, without having to modify or recompile the UUCP sources.

The rest of this article describes the commands that you can embed within **config**, should you wish to change the defaults for UUCP on your COHERENT system.

Miscellaneous Commands

The following **config** commands perform miscellaneous actions:

hdb-files true|false
If true, use HoneyDanBer configuration files instead of Taylor configuration files. COHERENT by default uses Taylor configuration files.

lockdir *directory*
Write lock files into *directory*. Under COHERENT, these files are written into **/usr/lib/uucp**.

max-uuxqts *number*
Set to *number* the maximum number of **uuxqt** processes that can run at any given time. The default is zero, which means that there is no limit.

nodename *name*
hostname *name*
uuname *name*
These commands are synonyms. Each tells UUCP to use *name* as the name of your system. Under COHERENT, your system's name is set in file **/etc/uuname**, and is returned by the system call **uname()**.

pubdir *directory*
Use *directory* as the publically accessible directory. Under COHERENT, the default public directory is **/usr/spool/uucppublic**.

run-uuxqt *string|number*
Specify when **uucico** should invoke **uuxqt**. If its argument is a number, **uucico** invokes **uuxqt** after it has received *number* execution files. If it is not a number, it must be one of the following strings:

once Invoke **uuxqt** once at the end of execution.
percall Invoke **uuxqt** once per call.
never Never invoke **uuxqt**.

Under COHERENT, the default is **once**.

spool *directory*
Use *directory* as the spool directory. Under COHERENT, the default spool directory is **/usr/spool/uucp**.

timetable *period time_string*
Define a time table to be used by default with subsequent **time** instructions. *period* is the period of day to which the time table applies. *time_string* is a standard time string that applies to that time of day. Taylor UUCP defines the following time tables by default:

```
timetable Evening Wk1705-0755,Sa,Su
timetable Night Wk2305-0755,Sa,Su2305-1655
timetable NonPeak Wk1805-0655,Sa,Su
```

unknown *string ...*
Let unknown systems log into your system. An "unknown," is one that is not described in **/usr/lib/uucp/sys**. Each *string* is applied to the unknown system, just as if it were named in **sys**. The COHERENT configuration of Taylor UUCP does not permit unknown systems to log in.

v2-files true|false
If true, use **V2**-style configuration files. COHERENT by default uses Taylor configuration files.

Configuration File Names

The following commands instruct Taylor UUCP to use configuration files other than the default ones:

callfile *file ...*
When dialing out, read the system name and password that your system passes to the remote system from each *file*. Taylor UUCP reads these files should the password or system name in a given system's description be set to '*'. Each line within a call file consists of three fields: the name of the remote system, the name by which your system identifies itself to the remote system, and the password. This mechanism permits you to make file **/usr/lib/uucp/sys** publically readable, while keeping the system names and passwords confidential.

COHERENT's default implementation of Taylor UUCP does not use call files, but you can set them up easily enough. Note that if you do so, pay careful attention to the permissions that you give each *file*.

dialcode *file ...*
Read dial codes from each *file*. "Dial codes" permits UUCP to interpret telephone numbers so they can be used through different telephone systems or area codes. COHERENT by default does not name or configure any dial-code files.

dialfile *file ...*
Read dialer-configuration information from every *file* instead of from the default file, **/usr/lib/uucp/dial**.

passwdfile *file ...*
Tell **uucico** to read system passwords from each *file*. This applies only to systems that are logging into your system, and only when **uucico** is managing the login process instead of the standard COHERENT programs. Each line in a *file* consists of two fields: the login name used by the remote system, and its password. **uucico** reads each file until it finds a password for the system that is attempting to log in.

Note that the COHERENT configuration of Taylor UUCP does not support encrypted passwords.

portfile *file ...*
Read port-configuration information from every *file* instead of from the default file, **/usr/lib/uucp/port**.

sysfile *file ...*
Read system-configuration information from every *file* instead of from the default file, **/usr/lib/uucp/sys**.

Log Files

The following commands let you change the log files that Taylor UUCP uses by default:

debugfile *file*
Write debugging information into *file* instead into the default file. Because COHERENT's port of Taylor UUCP uses HoneyDanBer logging instead of Taylor logging, **uucico** ignores this command. Under COHERENT, Taylor UUCP writes debugging information into **/usr/spool/uucp/.Admin/audit.local**.

logfile *file*
Write logging data into *file*. COHERENT's port of Taylor UUCP uses HoneyDanBer logging by default, which means that each system has its own log file within directory **/usr/spool/uucp/.Log**.

statfile *file*
Write statistics information into *file* instead of into the default file, **/usr/spool/uucp/.Admin/xferstats**.

Levels of Debugging

The COHERENT port of Taylor UUCP has debugging compiled into it. As noted above, under COHERENT Taylor UUCP writes its debugging information into file **/usr/spool/uucp/.Admin/audit.local**. You can place the command **debug** into file **config** to set the level of debugging to use by default.

LEXICON

Please What the Taylor documentation calls a *level* of debugging really records information about a given *activity*. For example, the command **debug chat** tells Taylor UUCP to record information about all actions taken while executing a chat script — not just the problems that occur while a chat script is being executed.

The command **debug** recognizes the following commands:

abnormal
Log abnormal situations.

chat Log chat-script activities.

handshake
Log activities during handshaking with the remote system.

uucp-proto
Log activities that involve the UUCP session protocol.

proto Log activities that involve individual link protocols.

port Log activities that involve the communications port.

config Log activities that occur while reading the configuration files.

spooldir
Log activities in the spool directory.

execute
Log whenever a program is executed.

incoming
Log all incoming data.

outgoing
Log all outgoing data.

all Log all of the above.

You can name more than one activity with the **debug** command. If you have more than one activity, the items in the list of activities must be separated by a comma instead of white space; for example, command

```
debug chat,handshake
```

tells UUCP to log activities that occur during execution of the chat script and handshaking.

A form of the **debug** command lets you invoke activities by number from the above list. note that the order is significant: **abnormal** is activity number zero, and **all** activity 11. For example, command

```
debug 3
```

tells UUCP to log activities zero through three — that is, **abnormal** through **uucp-proto**.

Note, too, that the **debug** command in this file can be overridden by using command-line option **-x** with any UUCP command.

See Also

Administering COHERENT, dial, port, sys, UUCP

connect() — Sockets Function (libsocket)

Connect to a socket
#include <sys/types.h>
#include <sys/socket.h>
int connect(*socket***,** *name***,** *namelen***)**
int *socket***,** *namelen***; struct sockaddr** **name***;*

The function **connect()** establishes a connection for a socket.

socket is a file identifier that describes a socket possessed by the current process. It must have been returned by a call to **socket()**. If it is of type **SOCK_DGRAM**, **connect()** specifies the peer with which the socket is to be connected; this address is that to which datagrams are to be sent, and the only address from which datagrams are to be received. If, however, it is of type **SOCK_STREAM**, **connect()** attempts to connect it with another socket. The

other socket is identified by *name*, which points to the full path name of the file to which the other socket is bound. This connection must have been established by a call to function **bind()**. *namelen* gives the length, in bytes, of the file name to which *name* points.

As a rule, a socket of type **SOCK_STREAM** can successfully connect only once; however, those of type **SOCK_DGRAM** sockets can call **connect()** multiple times to change their association. Datagram sockets can dissolve the association by connecting to an invalid address, such as a null address.

If the connection or binding succeeds, **connect()** returns zero. If an error occurs, it returns -1 and sets **errno** to an appropriate value. The following lists the errors that can occur, by the value to which **connect()** sets **errno**:

EBADF *socket* is somehow invalid.

ENOTSOCK
socket references a file, not a socket.

EADDRNOTAVAIL
The address is not available on this machine.

EAFNOSUPPORT
Addresses in the specified address family cannot be used with *socket*.

EISCONN
socket is already connected to an address or socket.

ETIMEDOUT
connect() timed out without establishing a connection.

ECONNREFUSED
The attempt to connect was forcefully rejected.

ENETUNREACH
The network is not reachable from this host.

EADDRINUSE
The address is already in use.

EFAULT
name gives an illegal address.

EINPROGRESS
socket is non-blocking yet the connection cannot be completed immediately.

EALREADY
The socket is non-blocking and a previous call to **connect()** has not yet been completed.

Example

For an example of this function, see the Lexicon entry for **libsocket**.

See Also

accept(), getsockname(), libsocket, select(), socket()

console — Device Driver

Console device driver

/dev/console is the device driver for the console of a COHERENT system. It is currently assigned major device number 2 and minor device number 0.

/dev/console interprets escape sequences in console output to control output on the console monitor. These escape sequences include the sequences from ANSI 3.4-1977 and ANSI X3.64-1979 that deal with terminal control. Thus, they are similar to those used by the DEC VT-100 and VT-220 terminals.

Escape Sequences

In addition to the ASCII control characters BEL, BS, CR, FF, HT, LF, and VT, **/dev/console** recognizes a number of special sequences, each of which is introduced by the ASCII character ESC. You can type these on the keyboard, or write them in a file and invoke them by **cat**ing the file to the standard output.

The following gives the escape sequences that **/dev/console** recognizes. The text in parentheses gives the ANSI

mnemonic for this escape sequence. Note that in this table, **ESC** represents the ASCII character ESC (i.e., 0x1B). **CSI** stands for Control Sequence Introducer, which here consists of the character ESC followed by the character '[' (0x5B). Note, too, that this table inserts spaces between characters. This is simply for the sake of legibility; at present, no escape sequence can contain a literal space character.

ESC = Enter alternate keypad mode. This escape sequence is non-standard and is slated for removal; you should avoid embedding it in scripts or programs.

ESC > Exit alternate keypad mode. This escape sequence is non-standard and is slated for removal; you should avoid embedding it in scripts or programs.

ESC *n* Print the special graphics character *n*.

ESC 7 Save the current cursor position. This escape sequence is non-standard and is slated for removal; you should avoid embedding it in scripts or programs.

ESC 8 Restore the previously saved cursor position. This escape sequence is non-standard and is slated for removal; you should avoid embedding it in scripts or programs.

ESC D (IND, Index)
Move the cursor down one line without changing the column position. This command moves the scrolling region text up and inserts blank lines if required. Although this escape sequence now moves the cursor down, it may not do so in the future when COHERENT supports writing systems other than left-to-right, top-to-bottom. Furthermore, this control sequence has been marked for removal from future international standards. This escape sequence has been slated for removal; you should avoid embedding it in scripts or programs.

ESC E (NEL, Next Line)
Move the cursor to the first column of the next line. This command move the scrolling region down and inserts blank line if required.

ESC M (RI, Reverse Index)
Move the cursor up one line without changing column position. As with IND, the direction of motion depends on the writing system currently in use.

CSI *n* **@** (ICH, Insert Character)
Insert *n* characters at the current position (default, one).

CSI *n* **A** (CUU, Cursor Up)
Move the cursor up *n* rows (default, one). Stop at top of page.

CSI *n* **B** (CUD, Cursor Down)
Move the cursor down *n* rows (default, one). Stop at bottom edge of scrolling region.

CSI *n* **C** (CUF, Cursor Forward)
Move the cursor *n* columns forward (default, one). Stop at right bottom corner of scrolling region.

CSI *n* **D** (CUB, Cursor Backwards)
Move the cursor *n* columns backwards (default, one).

CSI *n* **E** (CNL, Cursor Next Line)
Move the cursor *n* rows down (default, one). Move scrolling region up and insert a blank line if required.

CSI *n* **F** (CPL, Cursor Preceding Line)
Move the cursor *n* rows up (default, one). Move the scrolling-region text down and insert a blank line if required.

CSI *n* **G** (CHA, Cursor Character Absolute)
Move the cursor to column *n* of the current line.

CSI *n* **;** *m* **H** (CUP, Cursor Position)
Move the cursor to column *m* of row *n*. Position is relative to the scrolling region.

CSI *n* **I** (CHT, Cursor Horizontal Tabulation)
Move the cursor *n* tabulation stops forward (default, one).

CSI *c* **J** (ED, Erase in Display)
Erase display, where *c* is one of the following characters:

- **0** Erase from cursor to end of screen.
- **1** Erase from beginning of screen to cursor.
- **2** Erase the entire screen.

CSI *c* **K** (EL, Erase in Line)
Erase line, where *c* is one of the following characters:

- **0** Erase from cursor to end of line.
- **1** Erase from beginning of line to cursor.

2 Erase entire line.

CSI *n* **L** (IL, Insert Line)
Insert *n* blanks lines (default, one).

CSI *n* **M** (DL, Delete Line)
Delete *n* lines (default, one).

CSI *c* **O** (EA, Erase in Area)
Erase scrolling region, where *c* is one of the following characters:

0 Erase from cursor to end of scrolling region.
1 Erase from beginning of scrolling region to cursor.
2 Erase entire scrolling region. Reposition cursor at top left corner of scrolling region.

CSI *n* **P** (DC, Delete Character)
Delete *n* characters at the current position (default, one).

CSI *n* **S** (SU, Scroll Up)
Scroll the characters in the scrolling region up by *n* lines (default, one). The bottom of the scrolling region is cleared to blanks.

CSI *n* **T** (SD, Scroll Down)
Scroll the characters in the scrolling region down *n* lines (default, one). The top line of the scrolling region is cleared to blanks.

CSI *n* **X** (ECH, Erase Character)
Erase *n* characters at the current position (default, one).

CSI *n* **Z** (CBT, Cursor Backward Tabulation)
Move the cursor backwards by *n* tabulation stops (default, one).

CSI *n* **‘** (HPA, Horizontal Position Absolute)
Move the cursor to column *n* of the current line.

CSI *n* **a** (HPR, Horizontal Position Relative)
Move the cursor forward (i.e., to the right) *n* columns in the current line.

CSI *n* **d** (VPA, Vertical Position Absolute)
Move the cursor to row *n* of the display.

CSI *n* **e** (VPR, Vertical Postition Relative)
Move the cursor down *n* rows.

CSI *n* **;** *m* **f** (HVP, Horizontal and Vertical Position)
Move the cursor to column *m* of row *n*.

CSI *s1* **;** ... *sN* **m** (SGR, Select Graphic Rendition)
Select graphics rendition on the terminal. This command takes one or more colon-separated parameters *s1* through *sN*, each of which is one of the following strings:

0 All attributes off.
1 Bold intensity.
4 Underlining on. On color terminals, underlining rendered as white characters on a red background, in compliance with UNIX practices.
5 Blink on.
7 Reverse video.
10 Select primary font (see notes, below).
11 Select first alternative font (see notes, below).
12 Select second alternative font (see notes, below).
30 Black foreground.
31 Red foreground.
32 Green foreground.
33 Brown foreground.
34 Blue foreground.
35 Magenta foreground.
36 Cyan foreground.
37 White foreground.
40 Black background.
41 Red background.
42 Green background.
43 Brown background.
44 Blue background.

45 Magenta background.
46 Cyan background.
47 White background.

For example, the following command sets the foreground color to cyan and the background to black:

```
echo '\033[36;40m'
```

The following codes are not standard, and are slated for modification. Do not embed these codes in scripts or programs:

In the default font (font 0), **/dev/console** ignores control characters other than BEL, BS, CR, ESC, FF, HT, LF and VT and prints all other ASCII characters.

In font 1, **/dev/console** prints all characters (including control characters), except ESC.

In font 2, **/dev/console** prints every character except ESC with the high bit toggled. This provides access to the IBM graphics character set using ordinary ASCII characters.

CSI *n* **;** *m* **r** Make rows *n* through *m* of the display into the scrolling region. This is not a standard control sequence. It implements functionality included in standard sequences, and will be removed from a future console driver that implements the standard sequence.

CSI *c* **v** Select cursor rendition, where *c* is one of the following characters:

0 Cursor visible.
1 Cursor invisible.

This is not a standard sequence. It implements functionality not provided by any standard sequence. Developers are cautioned that there is no truly portable equivalent (although on many systems positioning the cursor off the screen has the same effect).

CSI ? 4 h (SM, Set Mode)
Enable smooth scrolling. This eliminates "snow" from the screen, but slows down the speed at which the console scrolls. The mode selected by the private-use parameter **?4** is not a standard mode.

Note that the term "smooth" is somewhat misleading; it means that the driver waits for vertical retrace before it updates video memory. The reason for waiting for retrace was that the old CGA tubes were poorly designed — the CRT logic and the main CPU were allowed simultaneous access to the video memory, with the result that direct-memory screen writes often produced static (snow). Having code wait for vertical retrace obviates the problem, but it also slows down the screen.

CSI ? 4 l (RM, Reset Mode)
Disable smooth scrolling. This is the default. The mode selected by the private-use parameter **?4** is not a standard mode.

CSI ? 7 h (SM, Set Mode)
Enable wraparound. Typing past column 80 moves the cursor to the first column of the next line, scrolling if necessary. The mode selected by the private-use parameter **?7** is not a standard mode, but is mandated by iBCS2.

CSI ? 7 l (RM, Reset Mode)
Disable wraparound. The cursor will not move past column 80. This is useful if the screen is being used as a block-mode interface. The mode selected by the private-use parameter **?7** is not a standard mode, but is mandaetd by iBCS2.

CSI ? 8 h (SM, Set Mode)
Erase in the current foreground color.

CSI ? 8 l (RM, Reset Mode)
Erase in the original foreground color, even if the current mode is reverse video.

CSI ? 25 h (SM, Set Mode)
Enable line 25.

CSI ? 25 l (RM, Reset Mode)
Disable line 25.

CSI > 13 h (SM, Set Mode)
Enable the screen saver. This is not standard.

CSI > 13 l (RM, Reset Mode)
Disable the screen saver. This is not standard.

ESC ' (DMI, Disable Manual Input)
Disable manual input. Terminal "beeps" (outputs **<ctrl-G>**) when you press a key on the keyboard. Interrupt and quit signals are still passed to the terminal process. Input may be renabled via **ESC c** (power up reset) or **ESC b** (enable manual input).

ESC b (EMI, Enable Manual Input)
Enable keyboard input that has been disabled by **ESC '**.

ESC c (RIS, Reset to Initial State)
Reset to power-up configuration

ESC t Enter keypad-shifted mode. This is a non-standard sequence that conflicts with explicit provisions of the relevant standards. It will be removed from future versions of the console driver in favor of a sequence that does not conflict.

ESC u Exit keypad-shifted mode. This is a non-standard sequence that conflicts with explicit provisions of the relevant standards. It will be removed from future versions of the console driver in favor of a sequence that does not conflict.

Numeric Keypad

The following describes the sequences sent by the numeric keypad.

The keypad sends the following escape sequences:

Key 0 Send **CSI L**.
Key 1 Send **CSI F**.
Key 2 Send **CSI B**.
Key 3 Send **CSI G**.
Key 4 Send **CSI D**.
Key 5 Send **ESC 7**.
Key 6 Send **CSI C**.
Key 7 Send **CSI H**.
Key 8 Send **CSI A**.
Key 9 Send **CSI I**.
Key . Send ASCII DEL.

When the **<shift>** key is pressed or the **<num-lock>** key is set, the keypad sends the literal characters '0' through '9' and '.'. If the **<num-lock>** key is set, pressing **<shift>** restores the escape sequences shown above.

The escape sequence **ESC =** sets the alternate-keypad mode. In this mode, the keypad sends the following escape sequences when the **<num-lock>** key is not set:

Key 0 Send **ESC ? p**.
Key 1 Send **ESC ? q**.
Key 2 Send **ESC ? r**.
Key 3 Send **ESC ? s**.
Key 4 Send **ESC ? t**.
Key 5 Send **ESC ? u**.
Key 6 Send **ESC ? v**.
Key 7 Send **ESC ? w**.
Key 8 Send **ESC ? x**.
Key 9 Send **ESC ? y**.
Key . Send **Esc ? n**.

The escape sequence **ESC >** resets this mode.

Other Special Keys

The following gives the escape sequences sent by the keyboard's special keys:

<home> Send "cursor home" (**CSI H**).
<up> Send "cursor up" (**CSI A**).
<pg up> Send **CSI I**.
<left> Send "cursor left" (**CSI D**).
<right> Send "cursor right" (**CSI C**).
<end> Send **CSI F**. Note that this escape sequence does not do what users normally expect: to send cursor to bottom left of screen, send the escape sequence **CSI 24 H**.

<down> Send "cursor down" (**CSI B**).
<pg dn> Move cursor to previous page (**CSI G**).
<ins> Send **CSI L**. Note that this escape sequence does not do what users normally expect.
<del> Send ASCII DEL.
F1 Send **CSI M**.
F2 Send **CSI N**.
F3 Send **CSI O**.
F4 Send **CSI P**.
F5 Send **CSI Q**.
F6 Send **CSI R**.
F7 Send **CSI S**.
F8 Send **CSI T**.
F9 Send **CSI U**.
F10 Send **CSI V**.
<shift>F1 Send **CSI Y**.
<shift>F2 Send **CSI Z**.
<shift>F3 Send **CSI a**.
<shift>F4 Send **CSI b**.
<shift>F5 Send **CSI c**.
<shift>F6 Send **CSI d**.
<shift>F7 Send **CSI e**.
<shift>F8 Send **CSI f**.
<shift>F9 Send **CSI g**.
<shift>F10 Send **CSI h**.
<ctrl>F1 Send **CSI k**.
<ctrl>F2 Send **CSI l**.
<ctrl>F3 Send **CSI m**.
<ctrl>F4 Send **CSI n**.
<ctrl>F5 Send **CSI o**.
<ctrl>F6 Send **CSI p**.
<ctrl>F7 Send **CSI q**.
<ctrl>F8 Send **CSI r**.
<ctrl>F9 Send **CSI s**.
<ctrl>F10 Send **CSI t**.
<ctrl><shift>F1
Send **CSI w**.
<ctrl><shift>F2
Send **CSI x**.
<ctrl><shift>F3
Send **CSI y**.
<ctrl><shift>F4
Send **CSI z**.
<ctrl><shift>F5
CSI @.
<ctrl><shift>F6
CSI [.
<ctrl><shift>F7
**CSI **.
<ctrl><shift>F8
CSI].
<ctrl><shift>F9
CSI ^.
<ctrl><shift>F10
CSI _.
<alt>F1 Send **CSI 1 y**.
<alt>F2 Send **CSI 2 y**.
<alt>F3 Send **CSI 3 y**.
<alt>F4 Send **CSI 4 y**.
<alt>F5 Send **CSI 5 y**.
<alt>F6 Send **CSI 6 y**.

<alt>F7	Send **CSI 7 y**.
<alt>F8	Send **CSI 8 y**.
<alt>F9	Send **CSI 9 y**.
<alt>F10	Send **CSI 0 y**.
<esc>	Send ASCII ESC (0x1B).
<tab>	Send ASCII HT.
<ctrl>	When combined with 'A' through '_', send the corresponding ASCII control character; when combined with the (¢) key, send ASCII LF; when combined with the key **<backspace>**, send ASCII DEL ; when combined with **<alt>** and **<del>**, issue system reset.
<shift>	Change alphabetic keys from lower case to upper case. If the **<caps-lock>** is set, shift from upper case to lower case.
<alt>	When combined with **<ctrl>** and **<del>**, issue a system reset.
<backspace>	Send ASCII BS; when combined with **<ctrl>**, send ASCII DEL.
<return>	Send ASCII CR; when combined with **<ctrl>**, send ASCII LF.
*	Send ASCII '*'.
<caps-lock>	Toggle "caps lock" mode.
<num-lock>	Toggle the interpretation of the numeric keypad, as described above.
<scroll-lock>	Send **<ctrl-S>** and toggle the Scroll Lock LED.
-	Send '-'.
+	Send '+'.

Altering Console Configuration

To change the hardware configuration of your console (i.e., to switch from a monochrome to a color console, or modify your keyboard or configuration of virtual consoles), log in as the superuser **root** and type the following commands:

```
cd /etc/conf
console/mkdev
bin/idmkcoh -o /kernel_name
```

where *kernel_name* is what you wish to name the newly built kernel. When you reboot, invoke *kernel_name* in the usual manner and your new configuration will have been implemented.

The following tunable kernel variables affect the behavior of the console driver:

CON_BEEP_SPEC
: This tunable kernel parameter lets you toggle whether the console can beep. If you set it to zero, the console will not beep, no matter what. By default, this is set to one, which enables beeping.

SEP_SHIFT
: This tunable kernel variable permits each virtual-console session to have its own settings for the keyboard's shift keys. When this variable is set to one, you can have **<CAPS LOCK>** turned on in one screen and **<NUM LOCK>** in another, and the driver correctly remembers the proper shift state when you switch sessions. The default for this variable is zero — that is, the keyboard uses the same settings for the shift keys in every virtual-console session.

See Also

Administering COHERENT, ASCII, device drivers, virtual consoles

Notes

Under COHERENT release 4.2, the codes sent by the keys **F1** through **F10**, **<pg up>**, **<pg dn>**, **<ins>**, **<del>**, and **<end>** have changed from those sent under previous releases. This was done so that COHERENT can more closely conform to the standard expected by many third-party packages. If this presents a problem, you can use the COHERENT command **fnkey** to change the codes sent by the function.

If you are using the keyboard driver **vtnkb**, you can remap the keyboard and (within limits) change the codes sent by some keys. For details, see the Lexicon entry **vtnkb**.

Beginning with COHERENT release 4.2, the console uses a 25-line screen, rather than the 24-lines used in previous releases. This is to support the numerous third-party packages that assume a 25-line display. A variant form of the **termcap** and **terminfo** entries for **ansipc** returns the screen to 24 lines, should you need that feature.

Please note that as of this writing (March 1994), the sequences **CSI** *n m***; do not work, where** *n* **is between 50 and 57. This is being worked repaired.**

const — C Keyword

Qualify an identifier as not modifiable

The type qualifier **const** marks an object as being unmodifiable. An object declared as being **const** cannot be used on the left side of an assignment (an *lvalue*), or have its value modified in any way. Because of these restrictions, an implementation may place objects declared to be **const** into a read-only region of storage.

See Also

C keywords, volatile
ANSI Standard §6.5.3

const.h — Header File

Declare machine-dependent constants
#include <sys/const.h>

The header file **const.h** declares most machine-dependent constants. These are constants that change among the various machines for which the COHERENT system is available; an example is the clock speed of the processor.

See Also

header files, times()

Notes

This header file is obsolete and will be dropped from a future release of COHERENT. Its use is strongly discouraged.

continue — Command

Terminate current iteration of shell construct
continue [*n*]

The command **continue** helps to control the flow of commands given to the shell. When it is used without an 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, commands, for, ksh, sh, until, while

continue — C Keyword

Force next iteration of a loop

continue forces the next iteration of a **for**, **while**, or **do** loop. For example,

```
while ((foo = getchar()) != EOF) {
      if ((foo < 'a') || (foo > 'z'))
            continue;
      ...         /* do something */
}
```

forces the **while** loop to throw away everything except lower-case alphabetic characters.

See Also

C keywords, for, while
ANSI Standard, §6.6.6.2

controls — System Administration

Data base for the lp print spooler
/usr/spool/mlp/controls

The file **/usr/spool/mlp/controls** is the data base for the print spooler **lp**. The superuser **root** can modify this file, either with a text editor or (to a more limited extent) with the command **lpadmin**.

The format of **controls** is simple. Every blank line is ignored. All text after the pound sign '#' is also ignored; you

can use this feature to embed comments in the file. The rest of the file consists of commands, each of which has the format *command*=*arguments*.

The following describes the commands that you can embed in **controls**:

default=*printer*
: This command sets the default printer, that is, the printer on which jobs are printed when the user does not specify a printer on the **lp** command line.

docopies=*status*
: This command controls how **lp** prints multiple copies. If it is set to **on**, then multiple copies are generated by invoking the printer's control script for each time; if it is set to **off**, then multiple copies are printed by telling the control script to do it. The difference in the two methods is that the former gives you more accurate information about the status of the job. If you wish to print many copies and you want to monitor the job's progress, then set **docopies** to **on**.

feed=*status*
: The command **localfeed** tells **lp** whether to insert a formfeed character between printing jobs sent to printers other than local printers. Setting it to **on** tells **lp** to output a formfeed character; setting it to **off** (or deleting it) tells it not to do so.

localfeed=*status*
: The command **localfeed** tells **lp** whether to insert a formfeed character between printing jobs sent to a local printer. (A *local* printer is one plugged into the auxiliary port of a terminal.) Setting it to **on** tells **lp** to output a formfeed character; setting it to **off** (or deleting it) tells it not to do so.

logroll=*hours*
: This command sets the time, in hours, at which the log file **log** is renamed **log.o** and a fresh log file is begun. This is done so the log file does not grow without bounds. The default value is 168 hours (one week).

longlife=*hours*
: Set, in hours, the "life-expectancy" of a file with a lifetime of **L**. The default is three days (72 hours).

printer=*name*,*device*,*script*
: This command defines a printer. **lp** accesses a printer by its name; it cannot access a printer unless you name it in a **printer** command. *name* names the printer. You can name a printer anything you like, so long as it is one word. *device* names the device into which it is plugged. *script* names the file in directory **/usr/spool/mlp/backend** that tells how to massage the text being passed to the printer. You can write or modify each script in that directory, and name each script whatever you like. Note that one physical printer can have multiple names, each using a different script; and one script can be shared by multiple physical printers.

  The command

```
printer = linenlq, /dev/lpt2, pannlq
```

  names a printer **linelq** that is plugged into port **/dev/lpt2**, and whose input is filtered through the contents of script **/usr/spool/mlp/backend/pannlq**.

  The command

```
printer = linepr, /dev/lpt2, linepr
```

  names a printer **linepr** that is plugged into **/dev/lpt2**, and whose input is filtered through the contents of script **/usr/spool/mlp/backend/linepr**.

  Note that these examples both name the same physical device. They differ in the scripts they use to massage their input; this will be described in detail below.

  Finally, a **printer** can direct its output to any device, serial or parallel, even **/dev/null**. For example:

```
printer=disk,/dev/null,disk
```

  As will be shown below, the script **disk** writes its output into a temporary file, so you can examine it without wasting a piece of paper. The format of a printer-control script is described below.

You do not have to include a printer-control script in a **printer** command; if you do not include one, the printer daemon **lpsched** uses the command **cat** by default.

shortlife=*hours*
Set, in days, the "life-expectancy" of a file with a lifetime of **S**. The default is 48 hours (two days).

templife=*hours*
Set, in minutes, the "life-expectancy" of a file with a lifetime of **T**. The default is two hours.

Printer Control Scripts

A printer-control script massages the text being handed to a given printer. The printer daemon **lpsched** redirects the output of the script (and therefore, of every command within the script) to the device named on the appropriate **printer** command named in the file **/usr/spool/mlp/controls**.

For example, consider the command

```
printer = linenlq, /dev/lpt2, pannlq
```

This command names a printer **linenlq**, declares that it is plugged into port **/dev/lpq2**, and requests that **lpsched** massage input to the printer through script **/usr/spool/mlp/backend/pannlq**. When **lpsched** processes a request that is directed to printer **linelq**, it pipes the text of the job into script **pannlq**, and redirects the output of **pannlq** to device **/dev/lpt2**.

It is important to remember that a printer-control script is not restricted to a few commands that the spooler understands. Each is a true shell script that can use any or all COHERENT commands to process text. The limits of what a script can do are set only by your imagination.

Consider the following examples. In the discussion, above, of the command **printer**, two scripts were mentioned: **pannlq** and **linepr**. Both send their output to the same physical printer, but they process the input text in different ways. The following gives the contents of **linepr**:

```
# filter the input through pr
pr

# throw a page at the end
echo "\f\c"
```

This script filters its input through the COHERENT command **pr**, which paginates the text and puts a header on it. It then echoes a formfeed character, to force the printer to throw a blank page at the end of the job. As in other shell scripts, a pound sign '#' introduces a comment and blank lines are ignored.

The following gives the contents of script **pannlq**:

```
# turn on near-letter-quality printing
echo "\021\033n"

pr

# turn off near-letter-quality printing
echo "\021\033P"
```

This script resembles the first, except that it includes commands to echo the magic strings that turn on and turn off near-letter-quality printing on this printer. This is one small example of the flexibility you can employ in devising a script

As with other shell scripts, you can modify the behavior of a printer-control script by setting environmental variables. For example, consider the following variation on the script **linepr**:

```
if [ $HEADER ]; then
      pr -h "$HEADER"
else
      pr
fi

# throw a page at the end
echo "\f\c"
```

If you have exported the environmental variable **HEADER**, then this script prints it at the top of each page; otherwise, it prints the default header. You can use the same technique to do other work, such as force the printing of a banner page.

The **lp** spooler reserves for its own use the environmental variables **MLP_COPIES**, **MLP_FORMLEN**, **MLP_LIFE**, **MLP_PRIORITY**, **MLP_SPOOL**. Your scripts can also use these variables. For more information on what each does, see its entry in the Lexicon.

When **lpsched** uses a printer-control script, it passes it three arguments: respectively, the sequence number of the print job (which identifies the job uniquely); the name of the user; and the number of copies being printed. You can use this information to control the printing of output; for example, consider the following:

```
for i in ‘from 1 to $3‘
do
      pr -h "User $2 - Copy $i of $3"
done
echo "\f\c"
```

Note, too, that just as each physical printer can be accessed in different ways via different scripts, so too the same script can be used by multiple physical printers. If you had multiple Panasonic printers plugged into your system, you could use the above script with each of them to massage their input appropriately.

One last example. As noted above, the output of a printer-control script can be directed to any device, not just a port. (It can also be redirected to non-existent ports, so be careful when you enter your **print** commands.) You can use this feature to redirect formatted text into files or other interesting places. Consider the following **printer** command:

```
printer=disk,/dev/null,disk
```

This creates a “printer” named **disk**. The text filtered through file **disk** is redirected to **/dev/null**. The contents of script **disk** show what this device is up to:

```
tee /tmp/D$$
```

This script uses the COHERENT command **tee** to redirect its input both to the standard output (which in the case of printer **disk** is thrown away) and into a file in directory **tmp**. You can use this command to save input for further examination later.

This discussion just scratches the surface of what you can do with the **lp** print spooler and its control scripts. For more information, see the Lexicon entries for **printer** and **lp**.

See Also

Administering COHERENT, lp, lpadmin, MLP_COPIES, MLP_FORMLEN, MLP_LIFE, MLP_PRIORITY, MLP_SPOOL, printer

conv — Command

Numeric base converter
conv [*number*]

conv converts *number* to hexadecimal, decimal, octal, binary, and ASCII characters, and prints the results on the standard output. If no *number* is given, **conv** reads one number per line from the standard input until you type the end-of-file character **<ctrl-D>**.

number may be 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 carat ‘^’. For example, it prints **<ctrl-X>** as ‘^X’. **conv** prints “bad digit” if anything is wrong with the input.

See Also

bc, commands, conv, dd, od, units

Notes

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

core — System Administration

Format of a core-dump file
#include <sys/core.h>

When a process terminates abnormally because a it encounters an unrecoverable error or receives an asynchronous signal from another process, COHERENT tries to write a copy of its image in memory into a file called **core**. You can examine this file with the debugger **db** and other tools to try to determine what went wrong.

The structure **ch_info** appears at the head of a **core** file. The header file **core.h** defines it as follows:

```
struct  ch_info {
      unsigned short ch_magic;
      unsigned int ch_info_len;
};
```

Field **ch_magic** is always set to the constant **CORE_MAGIC**. This "magic" value signifies to COHERENT that this is a core file. Field **ch_info_len** gives a count of information bytes in the core file, including the **ch_info** structure itself.

If the value of **ch_info_len** exceeds the size of the **ch_info** structure, this indicates that data follow the **ch_info** structure. These data follow the **ch_info** structure, and are in the form of a **core_proc** structure. Header file **<sys/core.h>** defines this structure as follows:

```
struct core_proc {
      gregset_t cp_registers;
      int cp_signal_number;
      struct _fpstate cp_floating_point;
      dregset_t cp_debug_registers;
};
```

This substitutes for a dump of the **u** area, whose information is reserved for the kernel alone.

This is followed by an image of each process segment. The data for each segment consists of the following: a header, which is a structure of type **core_seg**; **cs_pathlen** bytes of data that give the path name of the file from the segment data originated; and **cs_dumped** bytes of core-image data.

core.h defines strucutre **core_seg** as follows:

```
struct core_seg {
       size_t cs_pathlen;                    /* length of path name */
       off_t cs_dumped;                      /* dumped size in bytes */
       caddr_t cs_base;                      /* virtual base address */
       off_t cs_size;                        /* full size in bytes */
       unsigned long cs_reserved[8];
};
```

The order of the segments is the text segment first (if it is present — usually it is omitted), followed by the data segment, and then the stack segment. The contents of the text segment can usually be identified from the program being debugged. The patchable kernel variable **DUMP_TEXT** allows the COHERENT kernel to dump text segments as well as data and stack segments.

Patchable kernel variable **DUMP_LIM** sets the maximum size of a segment within a **core** file. The system uses this limit to keep keep core files from getting out of hand.

See Also

Administering COHERENT, core.h, signal(), wait()

Diagnostics

COHERENT will not write **core** if that file already exists as a non-ordinary file or if there is more than one link to it. The 0200 bit in the status returned to the parent process by **wait()** indicates a successful dump.

For a list of signals that automatically trigger a core dump, see the Lexicon entry for **signal()**.

core.h — Header File

Declare structure of a core file
#include <sys/core.h>

The header file **core.h** includes the structures and constants from which the system builds a **core** file. For more information on **core** files, see the Lexicon entry for **core**.

See Also

core, header files

cos() — Mathematics Function (libm)

Calculate cosine
#include <math.h>
double cos(*radian***) double** *radian***;**

cos() calculates the cosine of its argument *radian*, which must be in radian measure.

Example

For an example of this function, see the entry for **sin()**.

See Also

acos(), cosh(), libm
ANSI Standard, §7.5.2.5
POSIX Standard, §8.1

cosh() — Mathematics Function (libm)

Calculate hyperbolic cosine
#include <math.h>
double cosh(*radian***) double** *radian***;**

cosh() calculates the hyperbolic cosine of *radian*, which is in radian measure.

Example

The following example uses **cosh()** to compute the height and time to impact of a falling object. Assume that an object is acted on both by gravity and by air resistance propotional to v^2, where v is its velocity. When p is the proptionality constant for the resistance of air, the object's height after t seconds is given by the formula

```
y = y0 -1/p*ln(cosh(t*sqrt(p*g)))
```

and its time to reach the ground is given by the formula:

```
t = 1/sqrt(p*g)*log(exp(p*y0)+sqrt(exp(2*p*y0)-1))
```

Assuming that

```
g = 32 ft/s^2
```

the example computes an object's height after t seconds and the total time in seconds that it will take to reach the ground. It was written by Sanjay Lal (sanjayl@tor.comm.mot.com):

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

main ()
{
      float height, init_height, resistance, time_to_hit, g;
      int i;
      char buffer[50];

      g = 32.0;

      printf("Enter initial height, in feet: ");
      fflush(stdout);
      init_height = atof(gets(buffer));
```

```
      resistance = 0.0;
      while (resistance > 0.005 || resistance < 0.001) {
            printf("Enter air resistance (0.001 to 0.005): ");
            fflush(stdout);
            resistance = atof(gets(buffer));
      }

      time_to_hit = 1.0/sqrt(resistance*g) *
            log(exp(resistance*init_height) +
            sqrt(exp(2*resistance*init_height)-1));

      printf("Initial height: %1.0f\n", init_height);
      printf("Air resistance: %1.3f\n", resistance);
      printf("Time for object to hit the ground: %1.3f seconds\n",
            time_to_hit);

      /* countdown to impact */
      for (i = 2; ; i++) {
            height = init_height -
                  (1.0/resistance*log(cosh(sqrt(resistance*g)*(double)i)));

            if (height < 0) {
                  printf("BOOM!\n");
                  exit(EXIT_SUCCESS);
            } else
                  printf("Height after %i seconds: %1.3f feet\n", i, height );
      }
}
```

See Also

libm, cos()
ANSI Standard, §7.5.3.1
POSIX Standard, §8.1

Diagnostics

When overflow occurs, **cosh()** returns a huge value that has the same sign as the actual result.

cp — Command

Copy a file
cp [-d] *oldname newname*
cp [-d] *file1 ... fileN directory*

cp copies files. In its first form, **cp** copies the contents of *oldname* to *newname*, which it creates if necessary. If *newname* is a directory, **cp** copies *oldname* to a file of the same name in directory *newfile*.

In its second form, **cp** copies each *file*, from *file1* through *fileN*, into *directory.*

With the **-d** option, **cp** preserves the date (modification time) of the source file or files on the target file or files. By default, target files get the current time.

A file cannot be copied to itself.

See Also

commands, cpdir, ksh, mv, sh, wildcards

Notes

If you use **cp** to copy a file into another existing file, the newly copied file takes on the permissions of the file into which the text was copied. For example, consider the files **foo** and **bar**, whose permissions are as follows:

```
-rw-r--r--  1 fred       user            40 Tue Apr 14 18:19 bar
-rw-r-----  1 fred       user          1816 Tue Apr 14 20:53 foo
```

If you use **cp** to copy **foo** into **bar**, then typing **ls -l** shows the following:

```
-rw-r--r--  1 fred       user          1816 Tue Apr 14 21:37 bar
-rw-r-----  1 fred       user          1816 Tue Apr 14 20:53 foo
```

bar now has exactly the same contents as **foo** but retains its old set of permissions.

cpdir — Command

Copy directory hierarchy
cpdir [*option ...* **]** *dir1 dir2*

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 bits 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. This differs from the action of **cp**.

cpdir recognizes the following options:

-a Give a verbose account on one line of the files copied.

-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.

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

-t Test only, make no changes. With this option, **cpdir** 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 that would have been broken had the unlinking actions been executed.

-u Update regular files. Copy the source only if it was created or altered more recently than the target file, or if the target does not exist.

-v Print a verbose account of its activities. **cp** prints a file-by-file account of its actions, in addition to the information listed under **-t.**

See Also

commands, cp, link(), umask(), unlink()

cpio — Command

Archiving/backup utility

cpio is a standard utility that writes archives of files to disk or tape. Under COHERENT, **cpio** is a link to the command **gnucpio**. For details, see the Lexicon entry for that command.

See Also

commands, gnucpio

cpp — Command

C preprocessor
/lib/cpp [*option…***] [***file…***]**

The command **cpp** calls the C preprocessor to perform C preprocessing. It performs the operations described in section 3.8 of the ANSI Standard; these include file inclusion, conditional code selection, constant definition, and macro definition. See the entry on **C preprocessor** for a full description of C's preprocessing language.

Normally, **cpp** is used to preprocess C programs, but it can be used as a simple macro processor for other types of files as well. For example, the X utility **imake** uses **cpp** to help build makefiles.

cpp reads each input *file*, processes directives, and writes its product on **stdout**. If the option **-E** is not used, **cpp**

also writes into its output statements of the form **#line***n filename*, so the parser can connect its error messages and debugger output with the original line numbers in your source files.

Options

cpp recognizes the following options:

-C Do not suppress comments. Normally, **cpp** strips all comments from C code before it invokes the parsing phase, **cc0**.

-D*VARIABLE***[=***value***]**
Define *VARIABLE* for the preprocessor at compilation time. If *value* is not defined, *VARIABLE* is set to one. For example, the command

```
cc -DLIMIT=20 foo.c
```

tells the preprocessor to define the variable **LIMIT** to be 20. The C preprocessor acts as though the directive **#define LIMIT 20** were included in all source code.

-E Strip all line-number information from the source code. This option is used to preprocess assembly-language files or other sources, and should not be used with the other compiler phases.

-I*directory*
C allows two types of **#include** directives in a C program, i.e., **#include "file.h"** and **#include <file.h>**. The **-I** option tells **cpp** to search a specific directory for the files you have named in your **#include** directives, in addition to the directories that it searches by default. You can have more than one **-I** option on your **cpp** command line.

-o *file* Write output into *file*. If this option is missing, **cpp** writes its output onto **stdout**, which may be redirected.

-P Strip all file and line-number information from the C code. This is identical to the **-E** option, defined above.

-Q Suppress all messages.

-U*VARIABLE*
Undefine *VARIABLE*, as if an **#undef** directive were included in the source program. This is used to undefine the variables that **cpp** defines by default.

-V Print verbose messages.

-VCPLUS
Suppress C++-style online comments.

cpp reads the environmental variables **CPPHEAD** and **CPPTAIL** and appends their contents to, respectively, the beginning and the end of the command **cpp**.

See Also

C preprocessor, cc, commands

Diagnostics

The following gives the error messages returned by **cpp**. The messages are in alphabetical order. Each is marked as to whether it is a *fatal*, *error*, or *warning* condition. A fatal message usually indicates a condition that caused the compiler to terminate execution. Fatal errors from the later phases of compilation often cannot be fixed, and may indicate problems in the compiler or assembler. An error message points to a condition in the source code that the compiler cannot resolve. This almost always occurs when the program does something illegal, e.g., has unbalanced braces.

string argument mismatch *(error)*
The argument *string* does not match the type declared in the function's prototype. Either the function prototype or the argument should be changed.

#assert failure *(error)*
The condition being tested in a **#assert** statement has failed.

at beginning of macro *(error)*
Macro replacement lists may contain tokens that are separated by **##**, but **##** cannot appear at the beginning or the end of the list. The tokens on either side of the **##** are pasted together into one token.

at end of macro *(error)*
Macro replacement lists may contain tokens that are separated by **##**, but **##** cannot appear at the beginning or the end of the list. The tokens on either side of the **##** are pasted together into one token.

string: cannot create *(fatal)*
The preprocessor **cpp** cannot create the output file *string* that it was asked to create. This often is due to a problem with the output device; check and make sure that it is not full and that it is working correctly.

string: cannot open *(fatal)*
The compiler cannot open the file *string* of source code that it was asked to read. **cpp** may not have been told the correct directory in which this file is to be found; check that the file is located correctly, and that the **-I** options, if any, are correct.

cannot open include file *string (fatal)*
The program asked for file *string*, which was not found in the same directory as the source file, nor in the default **include** directory specified by the environmental variable **INCDIR**, nor in any of the directories named in **-I** options given to the **cc** command.

conditional stack overflow *(fatal)*
A series of **#if** expressions is nested so deeply that it overflowed the allotted stack space. You should simplify this code.

#define argument mismatch *(warning)*
The definition of an argument in a **#define** instruction does not match its subsequent use. One or the other should be changed.

#elif used without #if or #ifdef *(error)*
An **#elif** instruction must be preceded by an **#if**, **#ifdef**, or **#ifndef** control line.

#elif used after #else *(error)*
An **#elif** instruction cannot be preceded by an **#else** instruction.

#else used without #if or #ifdef *(error)*
An **#else** control line must be preceded by an **#if**, **#ifdef**, or **#ifndef** control line.

#endif used without #if or #ifdef *(error)*
An **#endif** instruction must be preceded by an **#if**, **#ifdef**, or **#ifndef** instruction.

EOF in comment *(fatal)*
Your source file appears to end in mid-comment. The file of source code may have been truncated, or you failed to close a comment; make sure that each open-comment symbol '/*' is balanced with a close-comment symbol '*/'.

EOF in macro *string* invocation *(error)*
Your source file appears to end in a macro call. The source file may be been truncated.

EOF in midline *(warning)*
Check to see that your source file has not been truncated accidentally.

EOF in string *(error)*
Your file appears to end in the middle of a quoted string literal. Check to see that your source file has not been truncated accidentally.

#error: *string (fatal)*
An **#error** control line has been expanded, printing the remaining tokens on the line and terminating the program.

error in #define syntax *(error)*
The syntax of a **#define** statement is incorrect. See the Lexicon entry for **#define** for more information.

error in #include syntax *(error)*
An **#include** directive must be followed by a string enclosed by either quotation marks (" ") or angle brackets (<>). Anything else is illegal.

identifier *string* has too many arguments *(error)*
Too many actual parameters have been provided.

illegal control line *(error)*
A '#' is followed by a word that the compiler does not recognize.

illegal cpp character (*n* decimal) *(error)*
The character noted cannot be processed by **cpp**. It may be a control character or a non-ASCII character.

illegal use of defined *(error)*
The construction **defined(***token***)** or **defined** *token* is legal only in **#if**, **#elif**, or **#assert** expressions.

string in #if *(error)*
A syntax error occurred in a **#if** declaration. *string* describes the error in detail.

include stack overflow *(fatal)*
A set of **#include** statements is nested so deeply that the allotted stack space cannot hold them. Examines the files for a loop. You should try to fold some of the header files into one, instead of having them call each other.

macro body too long *(fatal)*
The size of the macro in question exceeds the limit designed into the preprocessor. Try to shorten or split the macro.

macro expansion buffer overflow in *string (fatal)*
The COHERENT C compiler uses a static buffer space to expand preprocessor macros. In some extreme cases, a macro will exhaust this space, thus causing the C compiler to exit with this message. Try to shorten the macro, or break it up. See the Lexicon entry for **cpp** for suggestions on how to use an alternative C preprocessor to expand huge macros.

macro *string* redefined *(error)*
The program redefined the macro *string*.

macro *string* requires arguments *(error)*
The macro calls for arguments that the program has not supplied.

macros nested *number* deep, loop likely *(error)*
Macros call each other *number* times; you may have inadvertently created an infinite loop. Try to simplify the program.

missing #endif *(error)*
An **#if**, **#ifdef**, or **#ifndef** instruction was not closed with an **#endif** instruction.

missing output file *(fatal)*
The preprocessor **cpp** found a **-o** option that was not followed by a file name for the output file.

multiple #else's *(error)*
An **#if**, **#ifdef**, or **#ifndef** expression can be followed by no more than one **#else** expression.

nested comment *(warning)*
The comment introducer sequence '/*' has been detected within a comment. Comments do not nest.

new line in *string* literal *(error)*
A newline character appears in the middle of a string. If you wish to embed a newline within a string, use the character constant '\n'. If you wish to continue the string on a new line, insert a backslash '\' before the new line.

newline in macro argument *(warning)*
A macro argument contains a newline character. This may create trouble when the program is run.

out of space *(fatal)*
The compiler ran out of space while attempting to compile the program. To remove this error, examine your source and break up any functions that are extraordinarily large.

parameter must follow # *(error)*
Macro replacement lists may contain # followed by a macro parameter name. The macro argument is converted to a string literal.

preprocessor assertion failure *(warning)*
A **#assert** directive that was tested by the preprocessor **cpp** was found to be false.

string redefined *(error)*
: **cpp** macros should not be redefined. You should check to see that you are not **#include**ing two different versions of a file somehow, or attempting to use the same macro name for two different purposes.

too many arguments in a macro *(fatal)*
: The program uses more than the allowed ten arguments with a macro.

too many directories in include list *(fatal)*
: The program uses more than the allowed ten **#include** directories.

string: unknown option *(fatal)*
: The preprocessor **cpp** does not recognize the option *string*. Try re-typing the **cc** command line.

Notes

The COHERENT C compiler uses a static buffer space to expand preprocessor macros. Some programs that make especially intensive use of the C processor's macro facility may die during compilation with the message

```
macro expansion buffer overflow
```

This means that the program has exhaused the compiler's ability to process macros. You may wish to use an alternative preprocessor, such as the one that comes with **gcc**, as described below.

The COHERENT C compiler combines the preprocessor **cpp** with the parser **cc0**. The file **/lib/cpp** is simply a link to the C compiler **/lib/cc0**. Thus, there is no way to specify an alternative version of the preprocessor through the **cc** command. You can get around, this however, by linking the alternative preprocessor to a file named **cc0** in a directory other than **/lib**, then calling the alternative version via **cc**. For example, to have **gcc** preprocess program **hugemacro.c**, do the following. First, type the following commands to link the **gcc** preprocess to a file named **cc0**:

```
su root
cd /usr/local/lib/gcc-lib/i386-coh/2.3.2
ln cpp cc0
```

Then, to preprocess and compile **hugemacro.c**, type the following:

```
cc -t0 -B/usr/local/lib/gcc-lib/i386-coh/2.3.2 -E hugemacro.c > tmp.c
cc tmp.c
rm tmp.c
```

You may wish to embed the above into your **makefile**, or write it into a shell script.

CPPHEAD — Environmental Variable

Append options to beginning of cpp command line
export CPPHEAD=*options*

The COHERENT C preprocessor **cpp** reads the environmental variables **CPPHEAD** and **CPPTAIL** before it begins its work. You can set these variables to hold the default options that you want the preprocessor always to use.

cpp appends the options in **CPPHEAD** to the beginning of its command line.

See Also

cpp, CPPTAIL, environmental variables

CPPTAIL — Environmental Variable

Append options to end of cpp command line
export CPPTAIL=*options*

The COHERENT C preprocessor **cpp** reads the environmental variables **CPPHEAD** and **CPPTAIL** before it begins its work. You can set these variables to hold the default options that you want the preprocessor always to use.

cpp appends the options in **CPPTAIL** to the end of its command line.

See Also

cpp, CPPHEAD, environmental variables

creat() — System Call (libc)

Create/truncate a file
#include <fcntl.h>
int creat(*file***,** *mode***)**
char **file***; **int** *mode*;*

creat() creates a new *file* or truncates an existing *file*. It returns a file descriptor that identifies *file* for subsequent system calls. If *file* already exists, its contents are erased. In this case, **creat()** ignores the specified *mode*; the mode of the *file* remains unchanged. If *file* did not exist previously, **creat()** uses the *mode* argument to determine the mode of the new *file*. For a full definition of file modes, see **chmod()** or the header file **stat.h**. **creat()** masks the *mode* argument with the current **umask**, so it is common practice to create files with the maximal mode desirable.

Example

For an example of how to use this routine, see the entry for **open()**.

See Also

chmod(), fcntl.h, fopen(), libc, open(), stat.h, stdio.h
ANSI Standard, §4.9.3
POSIX Standard, §5.3.2

Diagnostics

If the call is successful, **creat()** returns a file descriptor. It returns -1 if it could not create the file, typically because of insufficient system resources or protection violations.

cron — System Administration

Execute commands periodically
/etc/cron&

cron is a daemon that executes commands at preset times.

Once each minute **cron** searches for commands to execute. **cron** first looks for file **/usr/lib/crontab**. If it exists, then **cron** reads it for commands to execute. If **/usr/lib/crontab** does not exist, however, **cron** searches **/usr/spool/cron/crontabs** for command files. Each user can have her own command file in that directory. See the Lexicon entry for **crontab** for information how to write and load a command file.

For each entry in each command file, **cron** compares the current time with the scheduled execution time and executes the command if the times match. When it finishes the search, **cron** sleeps until the next minute. Because it never exits, **cron** should be executed only once (customarily by **/etc/rc**).

cron is designed for commands that must be executed regularly. Temporal commands that need to be executed only once should be handled with the command **at**.

Permissions

cron performs some interesting manipulations with permissions. This is necessary to allow **cron** to run a wide variety of programs untended without creating loopholes in the system's security. Occasionally, this can create difficulties for users who do not grasp what **cron** does or why. The following describes how **cron** manipulates permissions on the programs you ask it to run.

To begin, when **cron** executes a user's **crontab** file, it changes the effective user ID to the ID of that user whose **crontab** file is being executed, **cd**'s to the user's **HOME** directory. When, however, **cron** runs an entry from a **/usr/lib/crontab**, it uses the user ID and group ID of user **daemon**. This prevents security holes involving entries in a **crontable** file.

For example, the following **crontab** entry contains redirection:

```
* * * * *   echo hello world >/dev/console 2>&1
```

If **cron** finds this entry in **/usr/lib/crontab**, it tries to execute the command as user **daemon**. The command will not execute it if user **daemon** lacks permission to write to **/dev/console**. Note that using **/usr/lib/crontab** is *not* recommended.

If however, it finds the entry in user **henry**'s **crontab** file, it tries to execute the command under the effective user ID of **henry**. The command will fail if **henry** lacks permission to write to **/dev/console**, and will succeed if he does.

When the shell executes a command in the background, it reads its standard input from **/dev/null** (unless redirected) and writes its standard output to the controlling tty. If **cron** is invoked with **/etc/cron&** from **/etc/rc**, there is no controlling tty, so the standard output goes to **/dev/null**. Thus,

```
* * * * *   echo hello world
```

typically writes **hello world** to **/dev/null**.

When a user logs in, **/bin/login** grabs the tty and runs **chown** and **chmod** on it. It is owned by the user with default permissions 700. If the user who has logged in on the console types the command

```
chmod /dev/console a+w
```

to allow all users to write to it, then the **crontab** entry

```
* * * * *   echo hello world >/dev/console 2>/tmp/cron.err
```

will indeed echo to the console every minute.

cron should be executed only once, at boot time. It uses **/usr/lib/cron/FIFO** as a lock file to prevent the execution more than one **cron** daemon.

If mail options are enabled, which is the default, **cron** sends mail to the owner of a **crontab** about all commands in that file that failed.

To allow **cron** to remove lock file **/usr/lib/cron**, do *not* send signal **KILL** to **cron**. Instead, use signal **TERM**. **cron** ignores signals **INT**, **HUP**, and **PIPE**. **cron** uses the signal **ALRM** internally.

Files and Directories

/usr/lib/cron/FIFO
Lock file (named pipe). Created by **cron**; removed by **cron/rc**.

/usr/lib/cron/cron.allow
List of allowed users. Permissions: **600 root root**.

/usr/lib/cron/cron.deny
List of denied users. Permissions: **600 root root**.

/usr/lib/crontab
Global **crontab** file, used by previous COHERENT **cron** mechanism.

/usr/spool/cron
Spool directory parent. Permissions: **700 root root**.

/usr/spool/cron/crontabs
Main **cron** directory. It holds each user's command file. Permissions: **700 root root**.

See Also

Administering COHERENT, commands, crontab

Notes

cron does not presently write into the log file. The size of the *hostname* + *domain* must not exceed 1,000 characters.

cron looks for **/usr/lib/crontab** to remain compatible with the COHERENT 286 version of **cron**. If, however, you continue to keep all **cron** commands in file **/usr/lib/crontab**, it will not be possible to run **setuid cron** tasks for logins that have a password. It is strongly recommended that you do *not* use **/usr/lib/crontab**, and instead create individual **crontab** files.

crontab — Command

Copy a command file into the crontab directory
/usr/bin/crontab [-l] [-r] [-f *filename***] [-m[ed]] [-u***user***]**

The command **crontab** copies a command file into directory **/usr/spool/cron/crontabs**. This directory holds the command files for all users. This mechanism permits each user to have her own file of commands to be executed periodically. If the file name is '-', then **crontab** reads the standard input.

crontab recognizes the following options.

-f *filename*
Replace your crontab file with *filename*.

-l List your crontab file.

-m[ed] Enable/disable the sending of mail to a user about any command in her crontab file that fails.

-r Remove your crontab file.

-u *user* Specify *user*. Only the superuser **root** can specify any user other than herself.

Allowing and Denying Access

The files **/usr/lib/cron/cron.allow** and **/usr/lib/cron/cron.deny** let the system administrator govern which users can use the **crontab** command:

- If **cron.allow** exists, then **crontab** checks its contents; if a given user is identified therein, then she can use **crontab**. Obviously, if **cron.allow** exists but is empty, then nobody can use **crontab**.
- If **cron.allow** does not exist, then **crontab** checks the contents of **cron.deny**. If a given user is identified therein, then she cannot use **crontab**; otherwise, she can. If **cron.allow** does not exist and **cron.deny** exists but is empty, then everyone can use **crontab**.
- If neither file exists, then everyone can use **crontab**.

Format of a crontab File

A **crontab** command file consists of lines separated by newlines. Each line consists of six fields separated by white space (tabs or blanks). The first five fields describe the scheduled execution time of the command. Respectively, they represent the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (0-6, 0 indicates Sunday). Each field can contain a single integer in the appropriate range, a pair of integers separated by a hyphen '-' (meaning all integers between the two, inclusive), an asterisk '*' (meaning all legal values), or a comma-separated list of the above forms. The remainder of the line gives the command to be executed at the given time.

For example, the **crontab** entry

```
29 * * 7 0 msg henry Succotash!
```

means that every hour on the half-hour during each Sunday in July, **cron** will invoke the command **msg**, and the user named **henry** will have the message

```
daemon: Succotash!
```

written on his terminal's screen (if he is logged in).

crond recognizes three special characters and escape sequences in a **crontab** file. If a command contains the percent character '%', **crond** executes only the portion up to the first '%' as a command, then passes the remainder to the command as its standard input. **crond** translates any percent characters '%' in the remainder to newlines. To prevent the special interpretation of a '%', precede it with a backslash, '\%'. Finally, **crond** removes the sequence **\<newline>** from the text before it passes the text to the shell **sh**; this can be used to make an entry in the **crontab** more legible.

You must pay special attention to permissions when you write a **crontab** command file. For information on how the **crontab** daemon **crond** manipulates permissions, see the entry for **crond** in the Lexicon.

Directories and Files

/usr/spool/cron/crontabs
Main **cron** directory. It holds each user's command file. Permissions: **700 root root**.

/usr/lib/cron/FIFO
Lock file (named pipe). Created by **cron**; removed by **crond/rc**.

/usr/lib/cron/cron.allow
List of allowed users. Permissions: **600 root root**.

/usr/lib/cron/cron.deny
List of denied users. Permissions: **600 root root**.

/usr/lib/crontab
Global **crontab** file, used by previous COHERENT **cron** mechanism. **/usr/spool/cron** Spool directory parent. Permissions: **700 root root**.

/usr/spool/cron/crontabs
Spool directory. Permissions: **700 root root**.

See Also

commands cron

Notes

COHERENT **crontab** is superset of the command of the same name included with UNIX System V release 3 (SVR3). The main differences are as follows:

- COHERENT **crontab** prints the usage when no options have been chosen, whereas SVR3 **crontab** reads stdin and can just remove the user's crontab file.
- SVR3 **crontab** does not include option **-f** *file_name*.
- SVR3 **crontab** does not include option **-u** *user*. Under SVR3 **crontab**, you must **su** to another user (e.g., **uucp**) before you can maintain her **crontab** file.

crypt — Command

Encrypt/decrypt text
crypt [*password***]**

The command **crypt** encrypts data. It emulates a rotor-encryption machine, such as the Enigma or Hagelin C-48 cipher machines. Unlike these machines, **crypt** uses 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 to standard output. *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 ten characters long, but must not be empty; all characters past the first ten 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, the commands

```
crypt COHERENT <file1 >file2
crypt COHERENT <file2 >file3
```

leave *file3* identical to *file1*.

Encrypted files produced by **ed** with its **-x** option may be read by **crypt**, and vice versa, as **ed** uses **crypt** to perform its encryption.

Security of a cryptosystem depends on several factors:

1. Brute-force attempts to break the system should be infeasible. Passwords should be at least five characters long; 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 less than four characters long).
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. Passwords must be kept secret. **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 superuser status, because the superuser can do practically anything. This is probably **crypt**'s most vulnerable point.

Files

/dev/tty — Typed passwords

See Also

commands, passwd, security, shadow

crypt() — General Function (libc)

Encryption using rotor algorithm
char *crypt(*key, extra***); char ****key***, ****extra***;**

crypt() implements a version of rotor encryption. It produces encrypted passwords that are verified by comparing the encrypted clear text against an original encryption.

key is an ASCII string that contains the user's password. *extra* is a "salt" string of two additional characters that are stored in the password file with the encrypted password. Each character must come from an alphabet of 64 symbols, which consists of the upper-case and lower-case letters, digits, the period '.', and the slash '/'.

crypt() returns a string built from the 64-character alphabet described above; the first two characters returned are the *extra* argument, and the rest contain the encrypted password.

See Also

libc

ct — Device Driver

Controlling terminal driver

Most processes that the COHERENT kernel executes are associated with a *controlling terminal*. (The only exceptions are daemon processes that are started by the process **init**.) This terminal directs I/O to the physical device through which the user who invoked the process is accessing COHERENT. Usually, this is a serial port or the console, but it could also be a socket (in the case of a **telnet** or **ftp** session), or some other device.

The driver **ct** lets a program access the controlling terminal automatically. It is accessed through the device **/dev/tty**. Thus, when a program invokes the system calls **open()**, **close()**, **ioctl()**, **read()**, or **write()** on **/dev/tty**, driver **ct** directs those calls automatically to the appropriate driver for the controlling terminal. This spares applications from having to know the details of the controlling device — all it has to do is manipulate **/dev/tty** and let **ct** take care of the details.

Files

/dev/tty

See Also

device drivers, init

Diagnostics

When a call finds no valid controlling terminal for a process, it returns a value of -1 and sets **errno** to **ENXIO.**

ctags — Command

Generate tags and refs files for vi editor
ctags [**-r**] *files*...

ctags generates the files **tags** and **refs** from a group of C-source files. **tags** is used by the **elvis** editor's **:tag** command, **<ctrl-]>** command, and **-t** option. **refs** is used by the command **ref**.

Each C-source file is scanned for **#define** statements and global function definitions. The name of the macro or function becomes the name of a tag. For each tag, a line is added to **tags**, which contains the following:

- the name of the tag
- a tab character
- the name of the file containing the tag
- a tab character
- a way to find the particular line within the file

refs is used by the command **ref**, which can be invoked via **elvis**'s **K** command. When **ctags** finds a global function definition, it copies the function header into **refs**. The first line is flush against the right margin, but the

argument definitions are indented. The command **ref** can search **refs** much faster than it could search all C-source files. The file-names list will typically include the names of all C-source files in the current directory, in the following format:

```
ctags -r *.[ch]
```

The **-r** to **ctags** tells it to generate both **tags** and **refs**. Without **-r**, it generates only **tags**.

See Also

commands, elvis, ref

Notes

This version of **ctags** does not parse ANSI source code very well. It has trouble recognizing the ANSI function definitions.

ctags is copyright © 1990 by Steve Kirkendall, and was written by Steve Kirkendall (kirkenda@cs.pdx.edu) assisted by numerous volunteers. It is freely redistributable, subject to the restrictions noted in included documentation. Source code for **ctags** is available through the Mark Williams bulletin board, USENET, and numerous other outlets.

Please note that this program is offered as a service to COHERENT users, but is not supported by Mark Williams Company. *Caveat utilitor.*

ctermid() — General Function (libc)

Name the terminal device that controls the current process
#include <stdio.h>
char *ctermid (*path_name***)**
char ****path_name*;**

The general function **ctermid()** returns the full path name of the terminal device that controls the current process. It does for the controlling terminal what the function **ttyname()** does for a general file descriptor.

path_name points to a block of memory into which **ctermid()** can write the name of the contolling terminal. It must point to at least **L_ctermid** bytes of available memory. If *path_name* is NULL, **ctermid()** writes the name into a statically allocated buffer that may be overwritten by subsequent calls to **ctermid()**.

If all goes well, **ctermid()** returns the address where it wrote the name of the controlling terminal. If an error occurs — for example, it could not discover the name of the controlling terminal — it returns an empty string.

See Also

libc
POSIX Standard 1003.1, §4.7.1

Notes

In almost every instance, **ctermid()** returns the string "/dev/tty". Under COHERENT, the name of the controlling terminal for the current process is **/dev/tty**. Because some operating systems do not follow this common practice, POSIX Standard provides **ctermid()** as a portable means of getting the controlling terminal's name.

ctime() — Time Function (libc)

Convert system time to an ASCII string
#include <time.h>
#include <sys/types.h>
char *ctime(*timep***)**
time_t ****timep*;**

ctime() converts the system's internal time into a string that can be read by humans. It takes a pointer to the internal time type **time_t**, which is defined in the header file **<sys/types.h>**, and returns a fixed-length string of the form:

```
Thu Mar  7 11:12:14 1989\n
```

ctime() is implemented as a call to **localtime()** followed by a call to **asctime()**.

Example

For another example of this function, see the entry for **asctime()**.

```
#include <time.h>
#include <sys/types.h>

main()
{
      time_t t;

      time(&t);
      printf("%s\n", ctime(&t));
}
```

See Also

libc, time [overview], time.h
ANSI Standard, §7.12.3.2
POSIX Standard, §8.1

Notes

ctime() returns a pointer to a statically allocated data area that is overwritten by successive calls.

ctype.h — Header File

Header file for data tests
#include <ctype.h>

ctype.h declares and defines the following routines, which can check and transform character types:

_tolower()	Convert an upper-case character to lower case
_toupper()	Convert a lower-case character to upper case
isalnum()	Test if alphanumeric character
isalpha()	Test if alphabetic character
isascii()	Test if ASCII character
iscntrl()	Test if a control character
isdigit()	Test if a numeric digit
isgraph()	Test if a graphics character
islower()	Test if lower-case character
isprint()	Test if printable character
ispunct()	Test if punctuation mark
isspace()	Test if a tab, space, or return
isupper()	Test if upper-case character
isxdigit()	Test if hexadecimal numeral
toascii()	Convert a character to ASCII
tolower()	Convert an upper-case character to lower case
toupper()	Convert a lower-case character to upper case

Example

The following example demonstrates **isalnum()**, **isalpha()**, **isascii()**, **iscntrl()**, **isdigit()**, **islower()**, **isprint()**, **ispunct()**, and **isspace()**. It prints information about the type of characters it contains.

```
#include <ctype.h>
#include <stdio.h>

main()
{
      FILE *fp;
      char fname[20];
      int ch;
      int alnum = 0;
      int alpha = 0;
      int allow = 0;
      int control = 0;
      int printable = 0;
      int punctuation = 0;
      int space = 0;
```

```
      printf("Enter name of text file to examine: ");
      fflush(stdout);
      gets(fname);

      if ((fp = fopen(fname, "r")) != NULL) {
            while ((ch = fgetc(fp)) != EOF) {

                  if (isascii(ch)) {
                        if (isalnum(ch))
                              alnum++;
                        if (isalpha(ch))
                              alpha++;
                        if (islower(ch))
                              allow++;
                        if (iscntrl(ch))
                              control++;
                        if (isprint(ch))
                              printable++;
                        if (ispunct(ch))
                              punctuation++;
                        if (isspace(ch))
                                    space++;

                  } else {
                        printf("%s is not ASCII.\n",
                                    fname);
                        exit(1);
                  }
            }

            printf("%s has the following:\n", fname);
            printf("%d alphanumeric characters\n", alnum);
            printf("%d alphabetic characters\n", alpha);
            printf("%d alphabetic lower-case characters\n",
                  allow);
            printf("%d control characters\n", control);
            printf("%d printable characters\n", printable);
            printf("%d punctuation marks\n", punctuation);
            printf("%d white space characters\n", space);
            exit(0);

      } else  {
            printf("Cannot open \"%s\".\n", fname);
            exit(2);
      }
}
```

See Also

header files, libc
ANSI Standard, §7.3

Notes

The argument for a **ctype** function or macro should be an **int** that is representable as an **unsigned char** or EOF — i.e., [-1, 0, ..., 255], as described in the ANSI standard §4.3.

The functions **_tolower()**, **_toupper()**, **isascii()**, and **toascii()** are not part of the ANSI standard. Programs that use them may not be portable to all implementations of C.

cu — Command

UNIX-compatible communications utility
cu [options] [*system***] [***phone***] [dir]**

The command **cu** implements a version of the communications utility used under UNIX System V. (Its name is an acronym for "call UNIX".) With it, you can interactively telephone other systems, upload files, download files, and perform other communications tasks. Unlike the program **ckermit**, which is also included with COHERENT, **cu** uses the information stored in UUCP data-base files **dial**, **port**, and **sys** to automate the dialing of a remote system.

To tell **cu** to dial a given system, just use that system's name on the **cu** command line. **cu** then reads from files

dial, **port**, and **sys** the information on how to dial the system you have named; then uses that information to open the port, set up the modem, and dial the system, up to the point where you see a login prompt on the remote system. For example, to dial system **mwcbbs**, use the command:

```
cu mwcbbs
```

Instead of dialing a remote system, you may wish to talk directly to a modem — for example, to reset its registers; or you may wish to log into a local system that is directly connected to your system via a serial port. To talk directly to a device, use the option **-p** followed by the name of the port into which the device is plugged, plus the command **dir**. This command tells **cu** that you wish to talk to the port directly. (Ports are named in the file **port**; for details, see its entry in the Lexicon.) For example, to talk directly a modem that is on a port named **MWCBBS**, use the command:

```
cu -p MWCBBS dir
```

To have **cu** dial a specific telephone number over a specific port, again use the option **-p** option to name the port, followed by the telephone number to call. For example, the command

```
cu -p MWCBBS 17085590412
```

connects to the modem on port **MWCBBS** and dial the telephone number 1-708-559-0412.

cu assumes that a string that begins with an alphabetic character names a system. To call a system whose name begins with numeral, use the command-line option **-z**, described below.

cu Commands

You can give commands to **cu** while you converse with the remote system. Each command begins with an escape character, which by default is the tilde '~'. **cu** recognizes the escape character only when it appears at the beginning of a line. After you type the escape character, **cu** replies with the name of your system, to show that it is ready to receive your command. If you do not see **cu**'s reply within a second or two, something has gone wrong.

To send to the remote system an escape character at the beginning of a line, enter it twice; for example, typing

```
~~
```

sends a single '~' to the remote system. All commands are either a single character or a word that begins with '%'.

cu recognizes the following commands:

~. Terminate the conversation.

~! *command*
Run *command* in a shell on your local system. If no *command* is given, start up a shell.

~$ *command*
Run *command* on your local system, and redirect to the remote system what *command* writes to the standard output.

~| *command*
Run *command* on your local system, and pipe into *command* what the remote system sends to your system.

~+ *command*
Combine the commands **~$** and **~|**. You can use this command to invoke alternative file-transfer utilities, e.g., **rz** and **sz**.

~#
~%break
Send a break signal.

~c *directory*
~%cd *directory*
cd to *directory* on your local system.

~> *file* Send *file* to the remote system. This command just dumps the file over the communication line, and performs no error checking. It assumes that the remote system is expecting it. You should first open a file on the remote system such as through the command

```
cat > filename
```

before you invoke this feature of **cu**.

~< Receive a file from the remote system. **cu** prompts you to name the file into which it will write what it receives from the remote system, then prompts you for the command to execute on the remote system to begin the file transfer (often, just **cat** *filename*). **cu** reads data from the remote system and writes them into into the file you named on your system until it detects the variable **eofread**.

~p *herefile farfile*
~%put *herefile farfile*
Copy (or *put*) file *herefile* on your system into file *farfile* on the remote system.

~t *farfile herefile*
~%take *farfile herefile*
Take file *farfile* from the remote system, and write it into file *herefile* on your system. This runs the appropriate commands on the remote system.

~s *variable* **[***value***]**
Set the **cu** *variable* to *value*. If no value is not given, set *variable* to **true**. **cu**'s variables are described below.

~! *variable*
Set the **cu** *variable* to **false**. **cu**'s variables are described below.

~%nostop
Turn off **XON/XOFF** flow control.

~%stop
Turn on **XON/XOFF** flow control.

~v List all **cu** variables and their values. **cu**'s variables are described below.

~? Help: list all **cu** commands.

cu Variables

The following variables are build into **cu** to control its default behaviors:

binary This variable indicates whether to pass binary information untouched when it transfers a file. If this variable is false, **cu** converts newline characters to carriage returns. If set to true, then **cu** passes binary data through untouched. The default is **false**.

binary-prefix
This variable gives the string that prefaces a binary character in a file transfer. This variable applies only if the variable **binary** variable is true. The default is **<ctrl-Z>**.

delay If this variable is true, **cu** delays for one second after it recognizes the escape character. The default is true.

echo-check
If **true**, **cu** checks file transfers by examining what the remote system echoes. This is not a robust method of checking the integrity of a transferred file, but it is the best that **cu** offers. The default is **false**.

echonl The character that **cu** looks for after it sends each line in a file. The default is the carriage return.

eofread This sets the string that **cu** looks for after it receives a file retrieved with the command **~<**. The default is **$**, which is intended to be a typical shell prompt.

eofwrite The string that **cu** writes after it sends a file with the command **~>**. The default is **<ctrl-D>**.

eol This variable gives the characters that **cu** recognizes as completing a line of input. **cu** recognizes the escape character only when it occurs immediately *after* one of the **eol** characters. **cu** recognizes the following **eol** characters by default: **<ctrl-C>**, **<ctrl-D>**, **<ctrl-O>**, **<ctrl-Q>**, **<ctrl-R>**, **<ctrl-S>**, and **<ctrl-U>**.

escape The escape character. By default, this is the tilde '~'.

kill This tells **cu** the character to use to delete a line if the echo-check fails. The default is **<ctrl-U>**.

resend The number of times to resend a line if the echo-check continues to fail. The default is ten.

timeout This variable sets the time, in seconds, that **cu** waits for a character either when it does echo-checking or when it looks for the **echonl** character. The default is **30**.

verbose Print accumulated information during a file transfer. The default is **true**.

To list the values of the variables, use the command **~v**. To modify a variable, use the commands **~s** or **~!**. For example, to turn off the one-second pause after sending an escape character, use the command:

```
~! delay
```

To change the escape character from '~' to '\', use the command:

```
~s escape \
```

Options

cu recognizes the following command-line options:

-a *port* The same as the option **-p**, described below.

-c *number* Dial *number*. You must use this option if the telephone number begins with a letter.

-d Enter debugging mode. This is equivalent to **-x all**.

-e Use even parity.

-*N* Equivalent to the command **-s** *N*, where *N* is an integer.

-h Half-duplex mode: echo locally all characters sent to the remote system.

-I *file* Use *file* instead of the configuration file.

-l *device* The device on which to dial out. Use this option to dial out on ports that are not list in the file **port**. You must have write permission on *device*.

-n Prompt for the telephone number to use.

-o Use odd parity. If you use both **-e** and **-o** on the command line, no parity is used. If neither is specified, **cu** uses the default parity of the line.

-p *port* The port to use. If you do not use this option, **cu** uses the default port for the system being contacted, as set in file **/usr/lib/uucp/sys**.

-s *speed* Set the baud rate to *speed*.

-t Map every carriage return character to the pair carriage/linefeed. Use this option when transferring files to an MS-DOS system.

-z *system* Call *system*. You must use this option if the name of the remote system begins with a numeral.

-x *activity* Log a given *activity*. These logs can help you debug problems with **cu**. **cu** recognizes the following activities:

abnormal
chat
handshake
port
config
incoming
outgoing

One **-x** option can name multiple activities, with the activities separated by commas. A **cu** command line can contain multiple **-x** options.

You can also use this option with a number, which turns on that many activities from the foregoing list, in the order in which the appear in this list. For example, the option **-x 2** is equivalent to the option **-x abnormal,chat** The option **-x all** logs on all activities.

See Also

ckermit, commands, dial, port, sys, UUCP

Notes

Unlike **ckermit**, the file-transfer facility in **cu** is primitive and performs no error checking. If you wish primarily to transfer files, you should consider using **ckermit** instead of **cu**. As noted above, the command **~+** plugs the standard input and standard output of two commands into each other; with this feature, you can use the other file-transfer utilities (e.g., **rz** and **sz**) to transfer files under **cu**.

cu requires that the device **/dev/console** appear last in file **/etc/ttys**. If this is not so, **cu** refuses to disable the enabled port or dial out. For details on this file, see the Lexicon entry for **ttys**.

cu was ported to COHERENT from the Taylor UUCP package, written by Ian Taylor (ian@airs.com).

curses.h — Header File

Define functions and macros in curses library
#include <curses.h>

curses.h defines the macros and declares the functions that comprise the **curses** library.

See Also

header files, libcurses, termcap, terminfo

cut — Command

Select portions of each line of its input
cut -c*list* **[***file ...***]**
cut -f*list* **[-s] [-d** *char***] [***file ...***]**

cut "cuts" one or pieces out of each line in its input, and writes the piece or pieces to the standard output. *list* specifies the pieces to cut out of each line. **cut** reads its input from *file*; if no *file* is named on its command line, **cut** reads the standard input.

A "piece" of an input line can be defined either as one or more characters from fixed positions in the line; or as one or more fields. The option **-c** selects characters from fixed positions; you would use this option if you were cutting up a file each of whose lines was of a fixed length. The option **-f** selects fields. A field does not have to have a fixed length, but its end must be marked by some special character; by default, a white-space character marks the end of a field. Option **-d** lets you specify the "magic character" that marks the end of a field. Option **-s** tells **cut** to throw away every line that does not contain the field-delimiter character. By default, **cut** will pass through unmodified every line that does not contain the field delimiter.

Options **-c** and **-f** are each followed by a *list*, which describes the pieces that you want from each input line. A piece is defined as follows:

N A piece consists of a single column or field. For example, the command

```
cut -f2 /etc/ttytype
```

selects field 2 from file **/etc/ttytype**.

N-N The range of columns or fields. For example, command

```
cut -c4-12 /etc/ttytype
```

selects columns 4 through 12, inclusive, from file **/etc/ttytype**.

-N Select every column or field from the beginning of the line through *N*. For example, command

```
cut -d\| -f-3
```

reads the first three fields from the standard input.

N- Select every column or field from *N* through the end of the line. For example, the command

```
cut -c15-
```

selects every character from character 15 through the end of the line.

If *list* defines more than one piece, the definitions of the pieces must be separated by commas. For example, the command

```
cut -c3-5,7-9
```

cuts columns three through five and seven through nine from the standard input, and writes them onto the standard output.

cut returns zero on success, one if an error occurred.

Examples

The following cuts column 4 through the end of the line from file **/etc/ttys**, and writes the cut piece onto the standard output. In effect, it throws away the first three columns of every line in that file:

```
cut -c4- /etc/ttys
```

You would use this command to display every serial-port device name that that file contains.

The next command selects fields one and six from file **/etc/passwd**. (Field one in this file gives a user's login identifier; and field six gives her home directory.) Note that fields in this file are delimited by a colon ':'.

```
cut -d: -f1,6 /etc/passwd
```

The final example cuts the first field from the input. It also explicitly sets the field delimiter to the space character. You would use this command to clip any trailing white space from data read from the standard input:

```
cut -f1 -d' '
```

See Also

awk, commands, paste, sed

Notes

cut is copyright © 1988,1990 by The Regents of the University of California. All rights reserved.

cvmail — Command

Convert mail from COHERENT 3.X format to SV format
cvmail [-m *filename***] [***filename***]**

The command **cvmail** converts to System V format existing COHERENT 3.X mailboxes and files used to store messages saved by COHERENT's 3.X mail utility.

To convert a default mailbox (i.e., a mailbox in directory **/usr/spool/mail**), invoke **cvmail** with its **-m** option, followed by the name of the user whose mailbox is to be converted. For example, to convert the mailbox belonging to user **bob**, type:

```
cvmail -m bob
```

If you have saved mail messages into a file, invoke **cvmail** with the name of the file to convert. For example, if you have stored mail messages in file **msg.save**, you can convert this file by typing:

```
cvmail msg.save
```

See Also

commands, mail

Notes

If you invoke **cvmail** without any arguments, it prompts you for the name of a file to convert. The file is not assumed to be a mailbox in directory **/usr/spool/mail**.

CWD — Environmental Variable

Current working directory

The Korn shell uses the environmental variable **CWD** to hold the current working directory.

See Also

environmental variables, ksh