d_passwd — System Administration

Give passwords for devices
/etc/d_passwd

The COHERENT system lets you force classes of users who log in through particular devices to enter an extra password. This helps you protect your system against people who may be try to break into your system via modem.

When a user attempts to log in, the command **login** check file **/etc/dialups** (should it exist) to see if this device is protected by an extra password. If this file names the device, **login** looks in file **/etc/d_passwd** to see if that user's shell is associated with an extra password. If that is so, then **login** prompts the user for that password, in addition to his usual password as set in file **/etc/passwd** or **/etc/shadow**.

Each entry in **/etc/d_passwd** has the following format:

> *shell***:***password***:***comment*

If field *shell* is empty, then **login** applies this password to all users who are using shells not named elsewhere within **d_passwd**.

The following gives an example of **d_passwd**:

```
/usr/lib/uucp/uucico::UUCP logins don't have extra password
/bin/sh:encrypted password:normal user with interactive shell
/usr/bin/ksh:encrypted password:normal user with interactive shell
```

To recreate the function of the account **remacc** (which **login** no longer recognizes as special), set **/etc/dialups** to name your dial-up ports, and set **d_passwd** to the following:

```
:encrypted_password:people/accounts dialing in
```

The following gives the contents of **d_passwd** from a typical COHERENT system:

```
:.03qn7EtBd.gi:Default dialup password
/usr/lib/uucp/uucico:.03qn7EtBd.gi:Dialup password for UUCP
/bin/sh:.03qn7EtBd.gi:Normal dialup extra password
/usr/bin/ksh:.03qn7EtBd.gi:Normal dialup extra password
```

The gibberish between the first and second ':' characters are the encrypted passwords. Note that this user has given the same password to each shell upon dialing up. This probably is a mistake.

See Also

Administering COHERENT, dialups, login

daemon — Definition

A *daemon* is a program that runs continually on your computer. It waits quietly for some condition to occur; then it awakens and performs some action (such as redirecting the file to a printer).

For example, the daemon **/etc/cron** wakes up every minute and checks every **cron** file. If a file contains a command to be executed at this time, then **cron** executes it.

As a general rule, anything that does not interact directly with users can be classified as a daemon. Daemons do not generally generate output to a user's terminal.

Any time you have a resource, like a printer or data base, to which access should be controlled, you can use a daemon.

For a list of daemons available under the COHERENT system, see the Lexicon entry for **Administering COHERENT**.

See Also

Using COHERENT

Notes

The function **bedaemon()**, which is included in **libmisc**, makes a program a daemon. See the article on **libmisc** for details.

A daemon may be killed accidentally, or through an error condition. When that occurs, a user may summon the daemon from the misty deep, but it will not come. The superuser **root** can reinvoke a daemon like any other program.

data formats — Definition

Mark Williams Company has written C compilers for a number of different computers. Each has a unique architecture and defines data formats in its own way.

The following table gives the sizes, in **char**s, of the data types as they are defined by various microprocessors.

Type	*i80386*	*i8086 SMALL*	*i8086 LARGE*	*Z8001*	*Z8002*	*68000*	*PDP11*	*VAX*
char	1	1	1	1	1	1	1	1
double	8	8	8	8	8	8	8	8
float	4	4	4	4	4	4	4	4
int	4	2	2	2	2	2	2	4
long	4	4	4	4	4	4	4	4
pointer	4	2	4	4	2	4	2	4
short	2	2	2	2	2	2	2	2

COHERENT places some alignment restrictions on data, which conform to all restrictions set by the microprocessor. Byte ordering is set by the microprocessor; see the Lexicon entry on **byte ordering** for more information.

Please note that Intel processor documentation and the Intel Binary Compatibility Standard (iBCS2) use the term *word* differently. The following table defines how they differ:

Bytes	*1*	*2*	*4*	*8*
Bits	*8*	*16*	*32*	*64*
Intel	**byte**	**word**	**dword**	**qword**
iBCS2	**byte**	**halfword**	**word**	**doubleword**

See Also

byte ordering, C language, data types, double, float, float.h, Programming COHERENT

Notes

COHERENT 286 supports Intel SMALL model only. COHERENT 386 supports the i80386 data format.

data types — Definition

COHERENT's implementation of C recognizes the following data types:

char
double
float
int
long
long float
long int
short
short int
signed char
signed int
signed long
signed long int
signed short
signed short int

unsigned int
unsigned long
unsigned long int
unsigned char
unsigned short
unsigned short int

The following types are synonymous:

char	**signed char**		
short	**short int**	**signed short**	**signed short int**
unsigned short	**unsigned short int**		
int	**signed int**		
long	**signed long**	**long int**	**signed lont int**
unsigned long	**unsigned long int**		
long float	**double**	**long double**	

The ANSI Standard states that **int**, **short**, and **long** are signed by default. It also sates that the implementation determines whether a **char** is signed or unsigned by default; but it does state that a printable character must be positive. COHERENT uses signed **char**s by default; therefore, if you wish to use a character value greater than 0x7F, you must explicitly declare the character to have type **unsigned char**. If you use this type in an arithmetic expression, COHERENT's C compiler automatically casts it to **unsigned int**.

Finally, COHERENT's header files define these commonly used data types:

<acct.h> **typedef unsigned short comp_t;**
<fcntl.h> **typedef struct flock flock_t;**
<signal.h> **typedef long sig_atomic_t;**
<stddef.h> **typedef int ptrdiff_t;**
<stdio.h> **typedef long fpos_t;**
<stdlib.h>
    typedef struct { int quot; int rem; } div_t;
    typedef struct { long quot; long rem; } ldiv_t;
<sys/acct.h>
    typedef unsigned short comp_t;
<sys/clist.h>
    typedef unsigned int cmap_t;
<sys/confinfo.h>
    typedef ddi_init_t init_t;
    typedef ddi_start_t start_t;
    typedef ddi_exit_t exit_t;
    typedef ddi_halt_t halt_t;
<sys/fd.h> **typedef unsigned fd_t;**
<sys/ksynch.h>
    typedef struct lock_info lkinfo_t;
    typedef struct sleep_lock sleep_t;
    typedef struct readwrite_lock rwlock_t;
<sys/mmu.h>
    typedef long cseg_t;
<sys/mzioctl.h>
    typedef long mzattr_t;
<sys/poll.h>
    typedef struct event event_t;
<sys/resource.h>
    typedef unsigned long rlim_t;
<sys/scsiwork.h>
    typedef struct scsi_work scsi_work_t;
    typedef struct scsi_cmd scsi_cmd_t;
<sys/seg.h>
    typedef long cseg_t;
<sys/signal.h>
    typedef n_sigset_t sigset_t;
    typedef o_sigset_t sigset_t;

<sys/stream.h>
    typedef struct free_rtn frtn_t;
<sys/types.h>
    typedef char *vaddr_t;
    typedef unsigned short minor_t;
    typedef unsigned short major_t;
<sys/uio.h>
    typedef struct uio uio_t;

See Also

C language, char, data formats, double, float, int, long, pointer, Programming COHERENT, short, unsigned

date — Command

Print/set the date and time
date [-s] [-u] [[*yymmdd***]***hhmm***[.***ss***]]**

date sets or prints prints the date and time of day.

If invoked without an argument, **date** prints the current date and time. It looks for the environmental variable **TIMEZONE**, which specifies local time zone and daylight saving time information. For details on the format of this variable, see the Lexicon entries for **TIMEZONE** and **ctime()**.

If invoked with a numeric argument (that is, one that consists of just digits, with no prefix), **date** interprets that argument as giving the current date and time, and uses it to set the current system time. The string must have the format *yymmddhhmm***[***ss***]**; the fields must be defined as follows:

yy	Year (00-99)
mm	Month (01-12)
dd	Day (01-31)
hh	Hour (00-23)
mm	Minute (00-59)
ss	Seconds (00-59)

For example, typing

```
date 940612141233
```

sets the date to June 12, 1994, and the time to 2:12:33 P.M. At least *hh* and *mm* must be specified — the rest are optional. **date** will complain and refuse to change the time should you attempt to set an impossible date or time, e.g., the date to February 30 or the time to 25 o'clock.

Note that the COHERENT command **ATclock** returns the date and time as recorded by your computer's internal clock. To reset the time as COHERENT understands it to the time as your computer understands it, use the command:

```
date `/etc/ATclock`
```

If you use option **-s** on **date**'s command line, **date** does *not* convert to daylight savings time when it sets the time.

If you use option **-u** on **date**'s command line, **date** sets and prints the date and time in Greenwich Mean Time (GMT) rather than in your local time.

See Also

ATclock, commands, ctime(), printf(), time, TIMEZONE

Notes

Only the superuser **root** can change the system's date or time.

The COHERENT version of the **date** command differs from the UNIX version in that the last two fields of its output are reversed. For example, the UNIX output of **date** reads

```
Sun Jan 13 12:02:09 CST 1991
```

where the COHERENT output reads:

```
Sun Jan 13 12:02:09 1991 CST
```

This may be important when importing UNIX shell commands into COHERENT.

db — Command

Assembler-level symbolic debugger
db [-a *symfile***] [-cdefort] [[***mapfile***]** *program***]**

db is an interactive, symbolic debugger. It allows you to run object files and executable programs under trace control (see the Lexicon entry for **ptrace**), run programs with embedded breakpoints, and dump and patch files in a variety of forms. You can use it to debug assembly-language programs that have been assembled by **as**, the Mark Williams assembler, and programs that have been compiled with the Mark Williams C compiler **cc**.

What is db?

db is a symbolic debugger, which means that it works with the symbol tables that the compiler builds into the object files it generates. Because **db** works on the level of assembly language, you need a working knowledge of i80386 assembly language and microprocessor architecture.

Invoking db

To invoke **db**, type its name, plus the options you want (if any) and the name of the files with which you will be working. *mapfile* is an object file that supplies a symbol table. *program* is the executable program to be debugged. If both names are given, the options default to **-c**. If only one name is given, it is the *program*; in this case the options default to **-o**. If both names are omitted, *mapfile* defaults to **l.out** or **a.out**, and *program* defaults to **core**. If possible, **db** accesses *program* with write permission.

db recognizes the following command-line options:

-a *symfile*
Read *symfile* for the list of symbols within the executable, instead of the executable's symbol table. This lets you copy an executable's symbol table in *symfile*, then strip that executable.

-c *program* is a core file produced by a user core dump. **db** checks the name of the command that invoked the process that produced the core, against the name of the *mapfile*, if given. Pure segments are read from the *mapfile*.

-d *program* is a system dump. If the command line names no files, *mapfile* defaults to **/COHERENT** and *program* defaults to **/dev/dump**.

-e The next argument is an object file; **db** executes it as a child process and passes it the rest of the command line. This permits the shell to expand wildcard characters that you place in the **db** command line, without spoiling the syntax of the **db** command.

-f Map *program* as a straight array of bytes (file).

-k Map program as a kernel process; *mapfile* defaults to **/coherent**, and *program* defaults to **/dev/kmem**.

-o *program* is an object file. If *mapfile* is given, it is another object file that provides the symbol table.

-p *prompt*
Change the command prompt from **db:** to *prompt*.

-r Only read the file, even if you have write permission for it. Use this to give a file additional protection.

-s Do not load symbol table.

-t Perform input and output for **db** via **/dev/tty**. This permits you to debug a process whose standard input or output has been redirected.

Commands and Addresses

db executes commands that you give it from the standard input. **db** displays the prompt

```
db:
```

when it is ready to receive a command. To change its prompt, use the **-p** option, described above. A command usually consists of an *address*, which tells **db** where in the program to execute the command; and then the command name and its options, if any.

An address is represented by an *expression*, which can be built out of one or more of the following elements:

- The '.', which represents the current address. When you enter an address, **db** sets the current address to that location. To advance the current address, type the **<Enter>** key.
- The name of a register. **db** recognizes the names of all registers on the 80386 microprocessor and the 80387 numeric co-processor. You can preceed a register name with a '%', but this is not required. If your program contains function **eax()**, the identifier **eax** identifies the function and **%eax** the register. If your program does not define **eax**, then either **eax** or **%eax** means the register. For example, the commands

```
sin:b
:e
:s
%st0?N
```

  sets a breakpoint at routine **sin()**, executes to it, single steps into it, and then prints the contents of the NDP stacktop **%st0**, which one step into **sin()** contains the argument.

  Typing the name of a register displays its contents. **db** displays register contents and stack traceback in hexadecimal values, regardless of the current default radix.
- The symbols **d**, **i**, and **u**, which represent location 0 in, respectively, the data space, the instruction space, and the u-area.
- The names of global symbols and symbolic addresses can be used in place of the addresses where they occur. This is useful when setting a breakpoint at the beginning of a subroutine.
- An integer constant, which can be used in the same manner as a global symbol. The default is hexadecimal; a leading **0** indicates octal and **0x** indicates hexadecimal.
- You can use the following binary operators:

+	Addition
-	Subtraction
*	Multiplication
/	Integer division

  All arithmetic is done in **long**s.
- You can use the following unary operators:

~	Complementation
-	Negation
*	Indirection

  All operators are supported with their normal level of precedence. You can use parentheses '()' for binding.

Every symbol refers to a segment: the data segment, the instruction segment, or the u-area. This segment, in turn, dictates the format in which **db** displays by default what it finds at that address. The format used by an expression is that of its leftmost operand. The symbols **d**, **i**, and **u** name specific segments in the absence of other symbols.

Displaying Information

To display information about *program*, use an expression of the form

[*address***][,***count***]?[***format***]**

This displays *format* for *count* iterations, starting at *address*. The symbol '.' represents the *address*, which defaults to the current display address if omitted. *count* defaults to one. The *format* string consists of one or more of the following characters:

^	Reset display address to '.'
+	Increment display address
-	Decrement display address
b	Byte
c	**char**; control and non-**char**s escaped
C	Like 'c' except '\0' not displayed
d	Decimal
f	**float**
F	**double**
i	Machine instruction, disassembled
l	**long**
n	Output '\n'
N	NDP (80387) register
O	octal
p	Symbolic address
s	String terminated by '\0', with escapes
S	String terminated by '\0', no escapes
u	**unsigned**
w	word
x	Hexadecimal
Y	time (as in i-node, etc.)

The format characters **d**, **o**, **u**, and **x**, specify a numeric base. Each of these can be followed by **b**, **l**, or **w**, which specify a datum size, to describe a single datum for display. A format item may also be preceded by a count that specifies how many times the item is to be applied. *format* defaults to the previously set format for the segment (initially **o** for data and u-area, and **i** for instructions). Except where otherwise noted, **db** increments the display address by the size of the datum displayed after each format item.

Execution Commands

In the following commands, *address* defaults to the address where execution stopped, unless otherwise specified; *count* and *expr* default to one. *commands* is an arbitrary string of **db** commands, terminated by a newline. A newline may be included by preceding it with a backslash '\'.

[*address***]=**
: Print *address* (offset) in hexadecimal. *address* defaults to '.'.

[*address***]=***value***[,***value***[,***value***]...]**
: Patch *value* into the program, beginning at point *address*. The address defaults to '.'. You can list up to ten *value*s. The command **=** assigns values to sequential locations in the traced process. **db** determines the size of the assigned value from the last display format used. You can set and display the registers of the traced process, just like any other address in the traced process.

? Print the last error message.

[*address***][,***n***]?[***ft***]**
: Display formatted information. *ft* indicates the format, which must be one of **bcCdfFilnNopsSuvwxY**. For details, see the command **:hf**, below

*address***?**
: Print *address*.

!*command*
: Pass *command* to a shell for execution.

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

[*address***]:b[***commands***]**
: Set a breakpoint at *address*. Execute *commands* when the breakpoint is encountered. *commands* defaults to **i+.:a\ni+.?i\n:x\n** — that is, print the breakpoint address, disassemble the instruction at the breakpoint address, and read more commands from the console.

:br [*commands***]**
: Set breakpoint at return from current routine, and execute *commands*. The default *commands* are the same as for **:b**, above.

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

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

[*address*]**:e**[*commandline*]
Begin traced execution of the object file at *address* (default, entry point). **db** parses *commandline* and passes it to the traced process. **argv[0]** must be typed directly after **:e** if supplied. For example,

```
:eprogname foo bar baz
```

sets **argv[0]** to **progname**, **argv[1]** to **foo**, **argv[2]** to **bar**, and **argv[3]** to **baz**. Quotation marks, apostrophes, and redirection are parsed as by the shell, but special characters '?*[]' and shell punctuation '(){}|;' are not. For complete shell command line parsing use the **-e** option, above.

Note that you must use the **:e** command to start the program execution prior to using the single-step, trace-back, or display-register commands. For example, the following COHERENT command sequence sets a breakpoint at **main()**, begins execution, and single-steps ten times through the program after having reached the breakpoint:

```
main:b
:e
,10:s
```

:f Print type of fault that caused a core dump or stopped the traced process.

:h Print help information.

:hf Print help information about display formats. **db** recognizes the following display formats:

b Byte.
c **char**; control, and non-**char**s printed as escape sequences.
C **char**; control and non-**char**s print as '.'.
d Decimal.
f **float**.
F **double**.
i Disassembled machine instruction.
l **long**.
n Output '\n'.
N NDP (80387) floating-point register (ten bytes).
o Octal.
p Symbolic address.
s String (NUL-terminated) with escape sequences.
S String (NUL-terminated).
u **unsigned**.
v File system l3-block address (three bytes).
w Word.
x Hexadecimal.
Y Time.

Options **d**, **o**, **u**, and **x** specify numeric bases (decimal, octal, unsigned decimal, hexadecimal). Each may be followed by **b**, **w**, or **l** to indicate a datum size (respectively, byte, word, or long).

:m Display segmentation map.

:p Display all breakpoints.

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

:r Display the contents of all registers on the microprocessor.

:rN Display the contents of all registers on the microprocessor and on the numeric co-processor. If your system does not possess a numeric co-processor, it displays the contents of the pseudo-registers used by COHERENT's emulator.

[*address*][,*count*]:s[c][*commands*]
Single-step execution starting at *address*, for *count* steps, executing *commands* at each step. *commands* defaults to **i+.?i**.

After a single-step command, **<Enter>** is equivalent to **.,1:s[c]**. The option **c** tells **db** to turn off single-stepping at a subroutine call and turn it on again upon return.

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

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

Note that the **:c**, **:s**, **:t**, and **:r** commands cannot be executed before a program is started. If you are debugging the program **hello**, do the following first:

```
db hello
main:b
:e
```

This invokes the debugger for **hello** and advances it to **main**. Now you can use the full set of commands.

Examples

The first example uses **db** to examine a program named **myprog**, which has core-dumped. To debug it, use the command

```
db myprog core
```

You could then issue the following commands to see where the problem lay:

:f This command displays the fault that caused the core dump.

:r This displays the contents of registers at the point where the program core dumped.

:t This command traces back the stack. With this command, you can see how your program arrived at the point where it core dumped. You can use this to find the point in your code where the program “jumps the rails”; often, this is all the information you need to fix the fault.

i1? This prints the value of global variable **i1** in your program at the time of the core dump.

:q Quit **db**. At this point, you should have a good idea of what went wrong with your program.

For another example, consider the following program, named **segv.c**:

```
main()
{
	register char *cp;

	cp = &main;
	*cp = 1000;
}
```

Compile this program with the command **cc segv.c**. To run it, type **segv**; as you can see, it crashes with a segmentation violation, producing a core-dump file named **core**. Now, you can use **db** to find out why the program core dumped.

To invoke the debugger, type:

```
db segv core
```

Now, type the **db** command:

```
:f
```

This tells **db** to print the type of fault that caused the program to dump core. **db** replies:

```
segmentation violation
```

Now, type:

```
*%eip?
```

db replies:

```
000000E9     movb (%ebx), $0xE8
```

Here, **db** gives you the value of the instruction pointer register **%eip** when the segmentation violation occurred and disassembles the instruction at that location. The offending instruction is trying to store indirectly through register **%ebx**. Type:

```
:t
```

db prints a traceback of the call stack:

```
7FFFFD24  000000E9  main(1, 0x7FFFFD38, 0x7FFFFD40)
```

This shows the program was in **main()** and not in any other function. Type:

```
:r
```

db prints contents of the machine registers:

```
%cs =000B       %eip=000000E9   %ss =0013       %fw =00011246
%ds =0013       %es =0013       %fs =0000       %gs =0000
%eax=00000001   %ebx=000000D4   %ecx=00000013   %edx=7FFFFD40
%esp=7FFFFD1C   %ebp=7FFFFD24   %edi=004090F4   %esi=00400D24
```

This shows that register **%ebx** has the value 0xD4 at the time of the core dump. Print the contents of **%ebx** symbolically:

```
%ebx?p
```

db replies:

```
00000020     main
```

The program is trying to store into the address of **main**. This causes a segmentation violation because COHERENT does not allow programs to write on code. Finally, type

```
:q
```

to exit from **db**.

In the last example, suppose you want to print the current address, the instruction at the current address, and the contents of global variable **j** when you hit function **fn** while running **db**. Type:

```
db cmd
main:b
:e
fn:b.:a\
.?i\
j?\
:x
```

The backslash '\' at the end of a line "escapes" a newline — that is, it tells **db** to ignore the newline, and concatenate the contents of the next line onto those of the present line. Thus, the **fn** command line (four physical lines with escaped newlines) forms a single **db** command that says the following:

.:a Print the current position as an address.
.?i Print the contents of the current position as an instruction.
j? print the contents of **j**.
:x Read more **db** input from the console.

The **:x** is necessary if you want to keep debugging interactively after **db** executes the breakpoint command list!

See Also

commands, coff.h, core, l.out.h, od, ptrace()

dbm.h — Header File

Header file for DBM routines
#include <dbm.h>

Header file **<dbm.h>** declares the functions used to manipulate DBM data bases:

dbmclose(). . . . . Close a DBM data base
dbminit(). . . . . Open a DBM data base
delete(). . . . . Delete a record from a DBM data base
fetch() . . . . Fetch a record from a DBM data base
firstkey(). . . . . Retrieve the first record from a DBM data base
nextkey(). . . . . Retrieve the next record from a DBM data base
store() . . . . Write a record into a DBM data base

It also defines the structure **datum**, which holds a data element, either a key or its associated data set:

```
typedef struct {
      char *dptr;
      int dsize;
} datum;
```

See Also

gdbm.h, header files, libgdbm, ndbm.h

Notes

Please note that function **dbmclose()** is non-standard. A program that uses it cannot be recompiled on an orthodox UNIX system.

For a statement of copyright and permissions on this header file, see the Lexicon entry for **libgdbm**.

dbm_clearerr() — NDBM Function (libgdbm)

Clear an error condition on an NDBM data base
#include <ndbm.h>
dbm_clearerr (*database***)**
DBM ****file*;**

Macro **dbm_clearerr()** clears an error that had been set on *database*.

See Also

Notes

As of this writing, this macro in fact does nothing.

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbm_close() — NDBM Function (libgdbm)

Close an NDBM data base
#include <ndbm.h>
void dbm_close (*database***)**
DBM ****database*;**

Function **dbm_close()** closes the NDBM data base to which *database* points. *database* must first have been opened by a call to **dbm_open()**.

See Also

Notes

This function is a wrapper for function **gdbm_close()**. It is included for compatibility with existing code.

If you have called **dbm_fetch()** to select data from *database*, you must use or copy the returned information before you call **dbm_close()**. If you do not, **dbm_close()** may corrupt the data in the **datum** that **dbm_fetch()** has returned.

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbm_delete() — NDBM Function (libgdbm)

Delete records from an NDBM data base
#include <ndbm.h>
int dbm_delete (*database***,** *key***)**
DBM ****database*;**
datum *key***;**

Database function **dbm_delete()** deletes the record with *key* from the data base to which *database* points. *database* must have been opened by a call to **dbm_open()**.

If all goes well, **dbm_delete()** returns zero. It returns -1 if *database* did not contain a record with *key*, or if *database* were opened into read-only mode.

See Also

Notes

This function is a wrapper for function **gdbm_delete()**. It is included for compatibility with existing code.

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbm_dirfno() — NDBM Function (libgdbm)

Return the file descriptor for an NDBM .dir file
#include <ndbm.h>
int dbm_dirfno (*database***)**
DBM ****database*;**

A NDBM data base consists of two files. One, with the suffix **.dir**, holds the index for the data base; the other, with the suffix **.pag**, holds the data themselves.

Function **dbm_dirfno()** returns the file descriptor for the **.dir** file associated with the data base to which *database* points. *database* must have been returned by a call to **dbm_open()**.

See Also

Notes

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbm_error() — NDBM Function (libgdbm)

Check a NDBM data base for an error
#include <ndbm.h>
int dbm_error (*database***)**
DBM ****database*;**

Macro **dbm_error()** checks *database* for an error condition, should a call to another data-base function fail.

See Also

Notes

Under the GDBM package, this macro in fact does nothing. It is included simply for compatibility with existing software.

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbm_fetch() — NDBM Function (libgdbm)

Fetch a record from an NDBM data base
#include <ndbm.h>
datum dbm_fetch (*database***,** *key***)**
DBM ****file*;**
datum *key***;**

Function **dbm_fetch()** retrieves from *database* the record with the given *key*. *database* must first have been opened through a call to function **dbm_open()**.

dbm_fetch() returns the address of the record it has retrieved. It returns NULL either if something went wrong (e.g., it could not read *database*), or if *database* does not contain a record with *key*.

See Also

Notes

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbm_firstkey() — NDBM Function (libgdbm)

Retrieve the first key from an NDBM data base
#include <ndbm.h>
datum dbm_firstkey (*database***)**
DBM **database***;**

Function **dbm_firstkey()** retrieves the record with the first key in *database*. *database* must first have been opened through a call to function **dbm_open()**.

dbm_firstkey() returns the address of the record it has retrieved. It returns NULL either if something went wrong (e.g., it could not read *database*), or if *database* is empty.

You can use **dbm_firstkey()** with function **dbm_nextkey()** to walk through *database*. For example:

```
for (key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

See Also

Notes

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbm_nextdbm() — NDBM Function (libgdbm)

Retrieve the next key from an NDBM data base
#include <ndbm.h>
datum dbm_nextkey (*database***)**
DBM **database***;**

Function **dbm_nextkey()** retrieves the next key from the data base to which *database* points. *database* must first have been opened via a call to function **dbm_open()**, and had the first key retrieved from it via a call to function **dbm_firstkey()**.

dbm_nextkey() returns the address of the record it has retrieved. If something has gone wrong, it returns NULL. If the last record within *database* has already been retrieved, it returns a record whose field **dptr** is NULL.

See Also

Notes

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbm_open() — NDBM Function (libgdbm)

Open an NDBM data base
#include <ndbm.h>
DBM **dbm_open (database***,** *type***,** *mode***)**
char **database***;**
int *type***,** *mode***;**

Function **dbm_open()** opens *database*. Parameters *type* and *mode* are the same as for the system call **open()**; for details, see its Lexicon entry.

To close *database*, call **dbm_close()**.

dbm_open() returns the address of the name of the data base it has opened. If something has gone wrong, it returns NULL.

See Also

Notes

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbm_pagfno() — NDBM Function (libgdbm)

Return the file descriptor for an NDBM .pag file
#include <ndbm.h>
int dbm_pagfno (*database***)**
DBM ****database*;**

A NDBM data base consists of two files. One, with the suffix **.dir**, holds the index for the data base; the other, with the suffix **.pag**, holds the data themselves.

Function **dbm_pagfno()** returns the file descriptor for the **.pag** file associated with the data base to which *database* points. *database* must have been returned by a call to **dbm_open()**.

See Also

Notes

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbm_rdonly() — NDBM Function (libgdbm)

Set an NDBM data base into read-only mode
#include <ndbm.h>
int dbm_rdonly (*database***)**
DBM ****database*;**

Function **dbm_rdonly()** puts an NDBM data base into read-only mode. *database* points to the data base; it must have been returned by a call to **dbm_open()**.

See Also

Notes

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbm_store() — NDBM Function (libgdbm)

Store a record into an NDBM data base
#include <ndbm.h>
int dbm_store (*database***,** *key***,** *content***,** *flags***)**
DBM ****database*;**
datum *key***,** *content***;**
int *flag***;**

Function **dbm_store()** inserts a record into *database*, which must first have been opened by a call to **dbm_open()**. *content* points to the data to be stored within the data base; and *key* points to the key under which the data are to be stored.

flag indicates how the the data are to be inserted into the data base. If it is set to **DBM_INSERT**, the data are appended onto the data base as new records; in this case, **dbm_store()** will not modify existing records that have an identical *key*. If, however, you set *flag* to **DBM_REPLACE**, *contents* replaces any existing record with an identical *key*.

If all goes well, **dbm_store()** returns zero. It returns a negative value should an error occur. If you set *flag* to **DBM_INSERT** and **dbm_store()** finds that *database* already contains a record with *key*, it returns one.

See Also

Notes

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libdbm**.

dbmclose() — DBM Function (libgdbm)

Close a DBM data base
#include <dbm.h>
void dbmclose()

Function **dbmclose()** closes a DBM-style data base. *database* points to the data base to be closed; it must have been returned by a call to function **dbminit()**.

See Also

Notes

This function is non-standard. A program that uses it cannot be recompiled on an orthodox UNIX system.

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dbminit() — DBM Function (libgdbm)

Open a DBM data base
#include <dbm.h>
int dbminit(*database***)**
char **database***;**

Function **dbminit()** opens and initializes a DBM data base. *database* points to the name of the data base to open.

Please note that unlike the GDBM function **gdbm_open()** or the DBM function **dbm_open()**, **dbminit()** does not create a data base — it merely opens it for manipulation. If the data base does not exist, you must first create it. To do so, create the empty files *database***.pag** and *database***.dir**.

If all goes well, **dbminit()** returns zero. If something goes wrong, it returns a negative value.

See Also

Notes

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

dc — Command

Desk calculator
dc [*file***]**

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

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

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

dc recognizes the following commands and constructions:

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

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

c Clear the stack.

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

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

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

I Stack the current assumed input base.

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

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

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

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

O Stack the current assumed output base.

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

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

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

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

x Remove the top of the stack, interpret it as a string containing a sequence of **dc** commands, and execute it.

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

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

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

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

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

Example

The following example program prints the first 20 Fibbonacci numbers. The character **l** is printed in boldface to help you tell from a numeric one.

```
1sa1sb1sc
[lalbdsa+psblc1+dsc21<y]sy
lyx
```

See Also

bc, commands

Notes

For most purposes, the in-fix notation of **bc** is more convenient than the Polish notation of **dc**.

dcheck — Command

Check directory consistency
dcheck [-s] [-i *inumber...***]** *filesystem ...*

dcheck checks the consistency of each *filesystem*. It scans all the directories in each *filesystem* and counts all i-nodes referenced. It then compares its counts against the link counts maintained in the i-nodes. **dcheck** notes any discrepancies, and notes allocated i-nodes with a link count of zero.

The **-i** argument tells **dcheck** to compare each *inumber* in the list against those in each directory. It reports matches by printing the i-number, the i-number of the parent directory, and the name of the entry. The **-s** argument tells **dcheck** to correct the link count of errant i-nodes to the entry count.

Because **dcheck** uses two passes to check a *filesystem*, the file system should be unmounted. If **-s** is used on the root file system, the system should be rebooted immediately (without performing a **sync**). The raw device should be used.

See Also

check, commands, icheck, ncheck, sync, umount

Diagnostics

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

Notes

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

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

dd — Command

Convert the contents of a file
dd [*option*=*value***] ...**

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

The following list gives each available *option*. Any numbers which specify block sizes or seek positions may be written in several ways. A number followed by **w**, **b**, or **k** is multiplied by two (for words), 512 (for blocks), or 1,024 (for kilobytes), respectively, to obtain the size in bytes. A pair of such numbers separated by **x** is multiplied together to produce the size. All buffer sizes default to 512 bytes if not specified.

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

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

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

ascii	Convert EBCDIC to ASCII
ebcdic	Convert ASCII to EBCDIC
ibm	Convert ASCII to EBCDIC, IBM flavor
lcase	Convert upper case to lower
noerror	Continue processing on I/O errors
swab	Swap every pair of bytes before output
sync	Pad input buffers with 0 bytes to size of **ibs**
ucase	Convert lower case to upper

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

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

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

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

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

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

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

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

Examples

The first example copies the entire contents of a 1.44-megabyte, 3.5-inch diskette from drive 0 to file **disk.dd**:

```
dd if=/dev/fva0 of=disk.dd bs=36b count=80
```

The second example writes the contents of the previously stored 5.25-inch file **backup.dd** to a 1.2-megabyte, 5.25-inch floppy disk in drive 1:

```
dd if=backup.dd of=/dev/fha1 bs=30b count=80
```

See Also

ASCII, commands, conv, cp, tape, tr

Diagnostics

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

Notes

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

decvax_d() — General Function (libc)

Convert a double from IEEE to DECVAX format
int
decvax_d(*ddp*, *idp***)**
double **ddp*, **idp***;**

decvax_d() converts a **double** from IEEE format to DECVAX format. *idp* points to the IEEE-format **double** to convert. *ddp* points to a destination for the converted DECVAX value; *ddp* may be identical to *idp* for in-place conversion.

decvax_d() returns zero on success, -1 on underflow, or one on overflow.

For a description of the IEEE and DECVAX formats for floating-point numbers, see the Lexicon article for **float**.

See Also

decvax_f(), float, ieee_d(), ieee_f(), libc,

decvax_f() — General Function (libc)

Convert a float from IEEE to DECVAX format
int
decvax_f(*dfp*, *ifp***)**
float **dfp*, **ifp***;**

decvax_f() converts a **float** from IEEE format to DECVAX format. *ifp* points to the IEEE-format **float** to convert. *dfp* points to a destination for the converted DECVAX value; *dfp* may be identical to *ifp* for in-place conversion.

decvax_f() returns zero on success, -1 on underflow, or one on overflow.

For a description of the IEEE and DECVAX formats for floating-point numbers, see the Lexicon article for **float**.

See Also

decvax_d(), float, ieee_d(), ieee_f(), libc

default — C Keyword

Default label in switch statement

default is a prefix used in **switch** statement. If none of the **case** labels match the parameter in the **switch** statement, then the **default** label is used. A **switch** is not required to have a **default** case, but it is good programming practice to use one.

See Also

C keywords, case, switch
ANSI Standard, §6.6.4.2

defined — Preprocessor Operator

Perform an action if a macro is defined

The preprocessor directive **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**.

See Also

#elif, #if, #ifdef, cpp, C preprocessor
ANSI Standard, §6.8.1

Notes

Note that **defined** is a preprocessor operator, not a preprocessor directive or a C keyword. The difference lies in the fact that you could write a function called **defined()** without any complaint from the C compiler; and if **defined** does not appear within an **#if** or **#elif** directive, the preprocessor ignores it.

deftty.h — Header File

Define default tty settings
#include <sys/deftty.h>

deftty.h defines the default tty settings.

See Also

header files

delete() — DBM Function (libgdbm)

Delete a record from a DBM data base
#include <dbm.h>
int delete (*key***)**
datum *key***;**

Function **delete()** deletes the record with *key* from the currently opened data base. The data base must first have been opened by a call to **dbminit()**.

If all goes well, **delete()** returns zero. If an error occurs, it returns a negative value.

See Also

Notes

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

deroff — Command

Remove text formatting control information
deroff [-w] [-x] [*file* **...]**

deroff removes text formatting control information from each input text *file*, or from the standard input if no *file* is specified. It regards all lines that begin with '.' or ''' as being **nroff** or **troff** commands and deletes them. **deroff**

also recognizes some additional control lines. It deletes **eqn** information (between **.EQ** and **.EN** lines), **tbl** information (between **.TS** and **.TE** lines), and macro definitions. It also deletes embedded **.eqn** requests. It expands source file inclusion with **.so** and **.nx** requests, with the proviso that no input file is read twice. It also deletes some **troff** escape sequences, such as those for font and size change.

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

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

See Also

commands, nroff, spell, troff, typo

detab — Command

Replace tab characters with spaces
detab [*tabsize***]**

The command **detab** reads the standard input, replaces every tab character with spaces, and writes the result to the standard output.

detab assumes that a tab stop occurs every *tabsize*, which must be an integer greater than one and less than 257. If you do not supply a *tabsize*, **detab** assumes that a tab stop occurs every eight characters. You can also override the default tab size by setting the environmental variable **TABSIZE** to a value other than eight.

See Also

commands

device drivers — Overview

A *device driver* is a program that controls the action of one of the physical devices attached to your computer system. The following table lists the device drivers included with the COHERENT system. The first field gives the device's major device number; the second gives its name; and the third describes it. If a major number does not appear in this table, that number is available for a driver yet to be written.

0:	**clock**	System clock
0:	**cmos**	System CMOS
0:	**freemem**	Amount of memory that is free at any given moment
0:	**idle**	System idle time
0:	**kmem**	Device to manage kernel memory
0:	**kmemhi**	
0:	**mem**	Interface to memory and null device
0:	**null**	The "bit bucket"
0:	**ps**	Processes currently being executed
1:	**ct**	Controlling terminal device (**/dev/tty**)
2:	**console**	Video module for console (**/dev/console**)
2:	**vtkb**	Non-configurable keyboard driver, virtual consoles
2:	**vtnkb**	Configurable keyboard driver, virtual consoles
2:	**mm**	The video driver
3:	**lp**	Parallel line printer
4:	**fd**	Floppy-disk drive
4:	**fdc**	765 diskette and floppy-tape controller
4:	**ft**	Floppy-tape drive
5:	**asy**	Serial driver
6:	**tr**	Trace driver
8:	**rm**	Dual RAM disk
9:	**pty**	Pseudoterminals
11:	**at**	AT hard disk
13:	**hai**	Host adapter-independent SCSI driver
14:	**cdu31**	Sony CD-ROM drives
16:	**mcd**	Mitsumi CD-ROM drives

Please note that the devices with major number 0 are not portable, and non-DDI/DKI. Also note that in future releases of COHERENT, the **hai** driver will be divided into several optional SCSI host-bus adapters (HBAs) and target devices.

It is not unusual for one major number to admit several driver service modules. Instances of this include the following major numbers:

0 This number is for a number of system-dependent drivers.

2 This number supports the console, both its keyboard modules and its video modules.

4 This describes varieties of floppy-disk and floppy-tape controllers and drives.

13 This describes a number of SCSI host modules, HBA modules, and target modules.

Major and Minor Numbers

COHERENT uses a system of *major* and *minor* device numbers to manage devices and drivers. In theory, COHERENT assigns a unique major number to each type of device, and a unique minor number to each instance of that type. In practice, however, a major number describes a device driver (rather than a device *per se*). The individual devices serviced by that driver are identified by a minor number. Sometimes, certain parts of the minor number specify configuration. For example, bits 0 through 6 of the minor number for COHERENT RAM disks indicate the size of the allocated device.

Optional Kernel Components

The kernel also contains the following optional components:

em87	Emulate hardware floating-point routines
msg	Perform System V-style message operations
sem	Perform System V-style semaphore operations
shm	Perform System V-style shared-memory operations
streams	Perform STREAMS operations

These components resemble device drivers, in that that they perform discreet work and can be linked into or excluded from the kernel, as shown below. However, they do not perform I/O with a device, and so are not true drivers. For details on these modules, see their entries in the Lexicon.

Configuring Drivers and the Kernel

Beginning with release 4.2, COHERENT lets you tune kernel and driver variables, enable or disable drivers, and easily build a new bootable kernel that incorporates your changes.

The command **idenable** lets you enable or disable a driver within the kernel. The command **idtune** lets you set a user-modifiable variable within the kernel. Finally, the command **idmkcoh** generates a new kernel that incorporates all changes you have made with the other three commands. Changes are entered with **idenable** and **idtune** do not take effect until you invoke **idmkcoh** to generate a new kernel, and boot the new kernel. Scripts **/etc/conf/*/mkdev** simpify the choices of **idenable** and **idtune** during installation and reconfiguration: they invoke **idtune** and **idenable** For details, see these commands' entries in the Lexicon.

Adding a New Device Driver

The commands described above make it easy for you to add a new device driver to your COHERENT kernel.

The following walks you through the processing of adding a new driver. We will add the driver **foo**, which enables the popular "widget" device. Please note that this example has the user modify the files **mtune** and **stune** by hand. It is not a good idea for you to do this; however, we describe how to do this to show how these files fit into the process of building a new kernel:

1. To begin, log in as the superuser **root**.
2. The next step is to create a directory to hold the driver's sources and object. Every driver must have its own directory under directory **/etc/conf**; and the sources must be held in directory **src** in that driver's directory. In this case, create directory **/etc/conf/foo**; then create directory **/et/conf/foo/src**.
3. Copy the sources for the driver into its source directory; in this case, copy them into **/etc/conf/foo/src**.
4. Create a **Makefile** in your driver's source directory, e.g., **/etc/conf/foo/src/makefile**. The easiest way to see what is required is to review several of the driver **Makefile**s shipped in the COHERENT driver kit. You can perform a test compilation of your driver by running **make** with the driver's **src** directory as the current

directory. This should create one object file that has the suffix **.o**. Copy this file in the driver's home directory, and name it **Driver.o**. In this case, the object for the driver should be in file **/etc/conf/foo/Driver.o**. In some rare cases, a driver compile into more than one object. You should store all of these objects into one archive; name the archive **Driver.a** and store it in the driver's home directory. The COHERENT commands that build the new kernel know how to handle archives correctly. The main idea is that files **Space.c** (if one exists) and **Driver.o** or **Driver.a** be placed in the driver directory, i.e., the parent of the **src** directory.

5. Add an entry to file **/etc/conf/sdevice** for this driver. **sdevice**, as described above, names the drivers to be included in the kernel. The entries for practically every entry are identical; you need to note only that the second column marks whether to include the driver in the kernel. In this case, the entry for the driver **foo** should read as follows:

```
foo   Y     0     0     0     0     0x0   0x0   0x00x0
```

For details on what each column means, read the comments in file **/etc/conf/sdevice**.

6. Add an entry to file **/etc/conf/mdevice** for the new driver. This file is a little more complex than **sdevice**; in particular, it distinguishes between STREAMS-style drivers and "old-style" COHERENT drivers. In most cases, you can simply copy an entry for an existing driver of the same type, and modify it slightly. In this case, the entry for **foo** should read as follows:

```
# full  func   misc   code    block  char   minor  minor  dma   cpu
# name  flags  flags  prefix  major  major  min    max    chan  id
foo     -      CGo    foo     15     15     0      255    -1    -1
```

In almost every case, the full name and the code prefix are identical. The code prefix also names the directory that holds the driver's object. Function flags are always always a hyphen, and miscellaneous flags almost always CGo. The block-major and character-major numbers again are almost always identical. The major number is usually assigned by the creator of the device driver. In future releases of the kernel, these will be assigned dynamically by the kernel itself; poorly written drivers that depend upon the driver having a magic major-device number will no longer work. Finally, the last four columns for non-STREAMS drivers are almost always 0, 255, -1, and -1, respectively. See the comments in file **/etc/conf/mdevice**.

7. If the driver has tunable variables, these should be set in the file **Space.c**, which should be stored in the driver's home directory. As it happens, **foo** does not need a **Space.c** file. For examples of such files, look in the various sub-directories of **/etc/conf**.

8. Type the command **idmkcoh** to build a new kernel. If necessary, move the new kernel into the root directory; you cannot boot it until it is in the root directory.

9. Save the old kernel and link the newly build kernel to **/autoboot**. You want save the old kernel, just in case the new one doesn't work. For directions on how to boot a kernel other than **/autoboot**, see the Lexicon entry for **booting**.

10. Back up your files! With a new driver in your kernel, it's best to play it safe.

11. Reboot your system to invoke the new kernel. If all goes well, you will now be enjoying the services of the new device driver.

For scripts on how to add or remove individual drivers from your kernel, see the article of the driver in question.

See Also

Administering COHERENT, asy, at, boot, console, ct, em87, floppy disk, hard disk, idle, kernel, lp, mboot, mdevice, mem, msg, mtune, null, pty, sdevice, sem, sgtty, shm, STREAMS, stty, stune, tape, termio

Notes

Note that in future releases of COHERENT, major numbers will not be static, as they are in the above table. Rather, they will be assigned by the **config** script when you install COHERENT onto your system. This scheme will allow more flexible arrangements of drivers, and will also allow COHERENT to support more than 32 drivers at once. If you write code to work with device drivers, you should *not* make any assumptions about a given driver's major or minor number.

See the Release Notes for your release of COHERENT for a full list of supported devices and device drivers.

Source code for almost all COHERENT device drivers is published in the COHERENT Device-Driver Kit. The only except is the source for **ft**, which includes proprietary information from manufacturers. Experienced writers of device drivers will find the driver kit a good tool for writing or importing drivers for devices that COHERENT does not yet support.

df — Command

Measure free space on disk
df [-fiv] [-t*filesys***]** (default format)

df measures the amount of space left free on the file system *filesys*. The file system being measured can reside on a hard disk, floppy disk, or RAM disk. For example, to check the amount of space left on file system **x**, type:

```
df /x
```

If you do not name a *filesys*, **df** prints information only about the file system that you in.

By default, **df** prints three statistics: the number of disk blocks free on this device, the total number of disk blocks in the device, and the percent of total disk blocks that is free. Note that a disk block is 512 bytes (1/2 kilobyte).

df recognizes the following command-line options:

-f Suppress i-node information.

-i Give the percentage of i-nodes available used.

-v Give the percentage of blocks used.

See Also

commands, mkfs

dial — System Administration

File that tells UUCP how to dial a system
/usr/lib/uucp/dial

The file **/usr/lib/uucp/dial** holds information about dialers. A *dialer* is a device, usually a modem, through which **uucico** or **cu** "dials" another computer system. The daemon **uucico** and the command **cu** use the information in this file to talk to dialers.

dial consists of a series of descriptions, each of which describes one dialer. A description consists of one or more commands; each command defines an aspect of how to manipulate the dialer. Descriptions must be separated by one blank line.

The following describes the commands you can use in a description:

dialer *name*
Name the dialer being described. Each description must begin with a **dialer** command. For example, the command

```
dialer trailblazer
```

introduces the description for the device named **trailblazer**. (A name need not be technical: you can also use names like **joe** or **junk_modem**.)

chat *from_modem to_modem ... from_modem*
This command gives the chat script with which **uucico** and **cu** initialize the dialer and have it dial a remote system. **chat** can have any number of arguments: the odd-numbered strings are received from the modem, and the even-numbered ones sent to it. Strings are separated by space character; therefore, no string can contain a literal space character. To represent a space character in a string, use the escape sequence **\s**.

If, at a given point in the conversation, nothing is expected from the modem or is to be sent to it, then use an empty pair of quotation marks as a placeholder.

Please note that unlike the chat script used in file **sys**, the chat script in **dial** contains only the information by which the modem is accessed: it does not contain information about how to log into the remote computer system.

A chat script can contain the following escape sequences:

\D Telephone number of the remote system
\T Telephone number plus dialcode translation
\M Do not require carrier
\m Require carrier, fail if not present
\s Represent a space character

uucico and **cu** use the command **phone** in file **/usr/lib/uucp/sys** to expand the escape sequence **\D**.

The following gives an example chat script:

```
chat  "" ATQ0V1E1L2M1DT\D CONNECT\s2400
```

The pair of quotation marks tells **uucico** (or **cu**) to expect nothing from the modem, and to send immediately the string **ATQ0V1E1L2M1DT** followed by the telephone number of the remote system. This is a typical send string for a Hayes-compatible, 2400-baud modem. The string also sets certain registers within the modem: **Q0V1** turns on verbal result codes, **E1** turns on echoing, and **L2M1** sets the duration and volume of the modem's speaker.

The last string in the chat script gives the *expect string*. This is the string that the modem sends when it has succeeded in connecting with the remote computer system. In this example, if the modem does not send

```
CONNECT 2400
```

then the attempt to call the remote system has failed. This example shows, as noted above, that no string to the command **chat** (or any other command used in **dial**) can contain a space character. To represent a space character within a string, use the escape sequence **\s**.

chat-timeout *seconds*
This command gives the number of seconds to await the expect string from the modem. For example, the command

```
chat-timeout 10
```

tells **uucico** to wait ten seconds for the expected string.

chat-fail *failure_string*
This command defines the string that, when received from the modem, indicates that a connection attempt has failed. **uucico** and **cu** abort when they receive *failure_string*. A dialer's description can have multiple **chat-fail** commands (after all, a call can fail for many different reasons). For example, the commands

```
chat-fail   BUSY
chat-fail   NO\sCARRIER
```

tell **uucico** and **cu** to abort when they receive either the strings **BUSY** or **NO CARRIER**.

chat-seven-bit true|false
If **true**, strip all bits to seven bits before comparing them with the expect string within the **chat** script.

chat-program *program* **[** *arguments* **]**
Run *program* before executing the chat script. The optional *arguments* are passed to *program*. The following escape sequences can be embedded within *arguments*:

\Y Name of the port device
\S Speed of the port
**** A literal backspace character

uucico expands these escape sequences before it passes *arguments* to *command*.

dialtone *stirng*
string is the code sequence that tells the modem to wait for a dial tone (e.g., if you must dial '9' and then pause briefly to get an outside line). **uucico** outputs *string* whenever it encounters a '=' within a telephone number. The default code is a comma.

pause *string*
string is the code sequence that tells the modem to pause for one second. **uucico** outputs *string* whenever it encounters a '-' within a telephone number. The default code is a comma.

carrier true | false
true indicates that the dialer supports the modem carrier signal, and **uucico** therefore will require that that carrier be on. **false** indicates that the dialer does not support the modem carrier signal, and **uucico** therefore will not wait for it.

carrier-wait *seconds*
Wait *seconds* for the carrier signal. The default is 60.

dtr-toggle true | false [true | false]
If the first argument is **true**, toggle DTR before using the modem. If the second argument is **true**, sleep for one second after toggling DTR.

complete-chat *string ... string*
complete-chat-timeout *number*
complete-chat-fail *failure_string*
complete-chat-seven-bit true | false
complete-chat-program *program* **[** *arguments* **]**
These commands define a chat script to be run after the UUCP session has run to completion. They are exactly like their **chat** counterparts described above.

abort-chat *string ... string*
abort-chat-timeout *number*
abort-chat-fail *failure_string*
abort-chat-seven-bit true | false
abort-chat-program *program* **[** *arguments* **]**
These commands define a chat script to be run if the UUCP session has aborted. They are exactly like their **chat** counterparts described above.

complete *string*
abort *string*
These are simplified of the **complete-** and **abort-** chat scripts described above. The former sends *string* to the dialer after a call has completed successfully; the latter sends its *string* after a call has aborted.

protocol-parameter *protocol parameter*
Set a protocol parameter. This command is exactly the same as its counterpart used in file **sys**. For details, see the Lexicon entry for **sys**.

seven-bit true | false
When your system negotiates the protocol to use with the remote system, force your system to accept only a protocol that works over seven-bit connection.

reliable true | false
When your system negotiates the protocol to use with the remote system, force your system to accept only a protocol that works over an unreliable connection.

half-duples true | false
If **true**, then the dialer supports only half-duplex connections. This forces your system to avoid bidirectional protocols during protocol negotiation.

Example

The following gives the entry for a 9600-baud Trailblazer modem:

```
dialer tbfast
chat "" AT\sE0\sQ4\sV1\sS7=60\sS50=255\sS51=255\sS66=0 \
      \sS111=30\sDP\D CONNECT\sFAST
chat-timeout 60
chat-fail BUSY
chat-fail NO\sCARRIER
chat-fail NO\sANSWER
abort-chat "" \d+++\dATH0\sV0\sE0\sQ1\sS0=1
abort-chat "" \d+++\dATH0\sV0\sE0\sQ1\sS0=1
```

Most of the commands in this example are optional. A dialer entry could work with only the first two commands. The following describes each command in detail:

dialer Give the dialer the name **tbfast**.

chat Give the chat script with which **uucico** converses with the modem. It sets a number of 'S' registers, turns echoing off, puts the modem into verbose mode, dials the remote system, and indicates that the signal for success is the string **CONNECT FAST**. Note that normally the chat script must be one unbroken string; this example is broken into two lines so it will fit onto the page. For information on the commands from which you would construct a chat script, see the documentation that comes with your modem.

chat-timeout Tells **uucico** how long to wait before it times out. In this case, wait 60 seconds.

chat-fail Define a string with which the modem indicates failure. In this case, there are three such commands, each naming a different message.

abort-chat
abort-chat These give the strings to send to the modem in the case of, respectively, the successful completion of call or an aborted call. For this entry, the same string is send in either case: it turns off echoing and verbose mode, and turns on auto-answering.

See Also

Administering COHERENT, port, sys, UUCP

Notes

Only the superuser **root** can edit **/usr/lib/uucp/dial**.

The file **dial** supports many commands in addition to the ones described here. This article describes only those commands that might be used in typical UUCP connections. For more information, see the original Taylor UUCP documentation, which is in the archive **/usr/src/alien/uudoc104.tar.Z**.

dialups — System Administration

Name every device that may need an additional password
/etc/dialups

The COHERENT system lets you force classes of users who log in through particular devices to enter an extra password. This helps you protect your system against people who may be try to penetrate it via modem.

The file **/etc/dialups** names every device that may require an additional password. Each device must be named on its own line; for example:

```
/dev/com1l
/dev/com3l
/dev/com3r
```

When a device is named in **/etc/dialups**, **login** looks in file **d_passwd** to see if a password has been linked to user's default shell. This permits you, for example, to ask for an extra password for all users who attempt to log in remotely and who have an interactive shell, while letting UUCP accounts enter without the extra password. For examples, see the Lexicon entry for **d_passwd**.

See Also

Administering COHERENT, d_passwd, login

diff — Command

Compare two files
diff [-bdefh] [-c *symbol***]** *file1 file2*

diff compares *file1* with *file2*, and prints a summary of the changes needed to turn *file1* into *file2*.

Two options involve input file specification. First, the standard input may be specified in place of a file by entering a hyphen '-' in place of *file1* or *file2*. Second, if *file1* is a directory, **diff** looks within that directory for a file that has the same name as *file2*, then compares *file2* with the file of the same name in directory *file1*.

The default output script has lines in the following format:

```
1,2 c 3,4
```

The numbers *1,2* refer to line ranges in *file1*, and *3,4* to ranges in *file2*. The range is abbreviated to a single number if the first number is the same as the second. The command *c* was chosen from among the **ed** commands '*a*', '*c*', and '*d*'. **diff** then prints the text from each of the two files. Text associated with *file1* is preceded by '< ', whereas text associated with *file2* is preceded by '>'.

The following summarizes **diff**'s options.

-b Ignore trailing blanks and treat more than one blank in an input line as a single blank. Spaces and tabs are considered to be blanks for this comparison.

-c *symbol*
Produce output suitable for the C preprocessor **cpp**; the output contains **#ifdef**, **#ifndef**, **#else**, and **#endif** lines. *symbol* is the string used to build the **#ifdef** statements. If you define *symbol* to the C preprocessor **cpp**, it will produce *file2* as its output; otherwise, it will produce *file1*. This option does *not* work for files that already contain **#ifdef**, **#ifndef**, **#else**, and **#endif** statements.

-e Create an **ed** script that will convert *file1* into *file2*.

-f Produce a script in the same manner as the **-e** option, but with line numbers taken directly from the two input files. This will work properly only if applied from end to beginning; it cannot be used directly by **ed**.

-h Compare large files that have a minimal number of differences. This option uses an algorithm that is not limited by file length, but may not discover all differences.

-d Select the **-h** algorithm only for files larger than 25,000 bytes; otherwise, use the normal algorithm.

Example

For an example of a script that uses this command, see the Lexicon entry for **trap**.

See Also

ed, egrep, commands, zdiff

Diagnostics

diff's exit status is zero when the files are identical, one when they are different, and two if a problem was encountered (e.g., could not open a file).

Notes

diff cannot handle files with more than 32,000 lines. Handing **diff** a file that exceeds that limit will cause it to fail, with unpredictable side effects.

diff3 — Command

Summarize differences among three files
diff3 [-ex3] *file1 file2 file3*

diff3 summarizes the differences among three text files. Each difference encountered is headed by one of the following separators, which categorizes how many of the three input files differ in a given range. The headers are as follows

==== All of the files are different.

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

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

*n: l***a** Text is to be appended after line *l* in *filen*.

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

Options are available to print a script of commands to **ed**. Option **-e** tells **diff3** to generate a script that makes all changes between *file2* and *file3* to *file1*. This script is based upon all changes flagged with the separators **====** or **====3**, as described above.

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

The option **-3** requests only those changes where *file3* differs.

Example

The following command sequence produces a script, applies it to *file1*, and sends the result to the standard output.

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

Files

/tmp/d3*
/usr/lib/diff3

See Also

commands, diff, ed

Diagnostics

An exit status of zero indicates all three files were identical, one indicates differences, and two indicates some other failure.

difftime() — Time Function (libc)

Calculate difference between two times
#include <time.h>
double difftime(*newtime***,** *oldtime***)**
time_t *newtime***,** *oldtime***;**

difftime() subtracts *oldtime* from *newtime*, and returns the difference in seconds. Both arguments are of type **time_t**, which is defined in the header **time.h**.

Example

This example uses **difftime()** to show an arbitrary time difference.

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

main()
{
      time_t      t1, t2;

      time(&t1);
      printf("Press enter when you feel like it.\n");
      getchar();
      time(&t2);

      printf("You waited %f seconds\n", difftime(t2, t1));
      return(EXIT_SUCCESS);
}
```

See Also

clock(), libc, mktime(), time [overview]
ANSI Standard, §7.12.2.2

directors — System Administration

Describe how to resolve local mail addresses
/usr/lib/mail/directors

The program **smail** reads file **/usr/lib/mail/directors** for the rules on how to resolve addresses on your local host. Please note that under COHERENT, the default configuration of **smail** does not use this file; however, if you wish, you can create it to change **smail**'s default rules for resolving local addresses.

Structure of Configuration Files

smail can use five varieties of configuration files:

- One or two *configuration* files, which perform global configuration of **smail**— including naming the other configuration files.
- One *directors* file, which describes how to deliver mail on your local system.

- One *routers* file, which describes resolve the addresses of remote systems.
- One *transports* file, which describes how to move mail from your system to selected remote systems.
- One *methods* file, which matches hosts with methods of transporting mail.

smail permits you to name these files as you choose; under COHERENT, they are named as follows:

/usr/lib/mail/config
/usr/lib/mail/directors
/usr/lib/mail/methods
/usr/lib/mail/routers
/usr/lib/mail/transports

Each is described in its own Lexicon entry. However, the **directors**, **routers**, and **transports** file all have the same format; the following describes it.

Each file consists of a set of entries; each entry, in turn, describes the attributes of one director, router, or transport. The order of entries in **director** and **router** is important, in that the directors and routers are invoked in the order stated in the file. The order of entries in the transport file is not important.

An entry in one of these files defines the following:

- A name by which that entry is known.
- A driver that implements the function for that entry.
- A set of generic attributes from a set that can be applied to any entry in the file.
- A set of driver-specific attributes, from a set that can be applied only to entries that use the specified driver.

For example, **directorsu** specifies the attributes for a director that reads aliases from a file **/private/usr/lib/aliases**:

```
# read aliases from a file private to one machine on the network
private_aliases:
      driver=aliasfile, owner=owner-$user ;
      file=/private/usr/lib/aliases
```

This entry is named **private_aliases**, and depends upon the low-level director driver routine named **aliasfile**. Errors found while processing addresses found by this director are sent to an address formed by prefixing the string **owner-** to the name of the alias; these aliases are stored in file **/private/usr/lib/aliases**. The director-driver **aliasfile** implements a general mechanism for looking up aliases stored in a data base. By default, aliases are kept in a DBM-style data base that is built from the text file **/usr/lib/mail/aliases**. For details on this file and its format, see the Lexicon entry for **aliases**. For details on how DBM-style data bases, see the Lexicon entry for **libgdbm**.

The separation between generic attributes and driver-specific attributes mirrors the internal design of **smail**. Above the driver level, routines exist that implement aspects of drivers, routers, and transports but do not depend upon the specific means for performing the operation. These higher-level functions can be manipulated through the generic attributes. On the other hand, the drivers that actually perform these operations accept a different set of attributes to control their behavior. In the case of a driver thats read or writes to a file, a file attribute usually exists. In the case of a driver that executes a program, a **cmd** attribute usually exists to specify how that program is to be invoked.

Attributes of a Director

The following the generic attributes can be used in an entry in **directors**. Each attribute is followed by its type (Boolean or string). To set a string attribute, its name should be followed by an '=', then the value to which you are setting it. To set a Boolean attribute, prefix it with a '+'; to unset a Boolean attribute, prefix it with a '-'.

caution (Boolean)
: If set, then be cautious of the addresses this director produces. If the attribute **nobody** is not set, then reject file, shell-command, or :include:filename-style mailing-list addresses.

default_group (string)
: If the driver does not associate a group to an address returned by it, then associate the group identifier for this group name. This will override the group identifier set by the attribute **default_user**.

default_home (string)
If the driver does not associate a home directory with an address returned by it, then use this directory as the default home directory. **smail** expands the value of this attribute to form the directory path name. At present, variable **$user** is not available for this expansion. If the string expansion fails, **smail** ignores it.

default_user (string)
If the driver does not associate a user or group to an address returned by it, then associate the user identifier and group identifier of this user.

driver (string)
This attribute names the set of low-level functions that do the work of directing local mail. This attribute is required.

nobody (Boolean)
If set, then **smail** accesses files or runs shell commands as the user specified by its attribute **nobody**, for addresses flagged with caution by either the caution generic attribute or by the driver. Association of **nobody** with an address overrides the attributes **default_user**, **default_group**, **set_user**, and **set_group**. This attribute is set by default.

owner (string)
This names the address to be sent mail if an error occurs while **smail** is processing the addresses produced by this director. This string is expanded with the variable **$user** set to the local-form address passed to the director. By deafault, the value **owner-$user**. If this string expansion fails, **smail** ignores it.

sender_okay (Boolean)
If set, then it is always okay for this attribute to produce an address equal to the sender. This effectively turns on the "me too" flag for this director. This should generally be set for forwarding directors and should not be set for aliasing and mailing-list directors.

set_group (string)
Associate this group's identifier with the addresses that the driver returns. This overrides any group identifier set by the attribute **set_user**.

set_home (string)
Associate this home directory with all addresses returned by the driver. This will be expanded in the same manner as **default_home**.

set_user (string)
Associate the user and group identifiers for this user with addresses that the driver returns. This overrides any values set by the driver.

smail requires that two addresses exist: **Postmaster** and **Mailer-Daemon**. To avoid the necessity of an alias for these two users, **smail** contains two implicit directors embedded into the directing code; it uses them as a last resort. The first such director maps the address **Mailer-Daemon** onto the address **Postmaster**; and the second maps **Postmaster** onto the address **root**.

The Preloaded Directors

If **smail** does not find a copy of file **directors** in directory **/usr/lib/mail** (which is the case by default under COHERENT), it uses its the default configuration. The default director configuration supports the following directors:

Include Files
smail expands local addresses of the form **:***include***:***filename* into a list of addresses contained in the ASCII file *filename*. The files to which these addresses refer are called *mailing list files*. This form of local address can appear in any alias file, forward file, or mailing-list file. A user cannot supply such an address himself.

Alias Files
This director scans for entries in an DBM-style data base that is built from text file **/usr/lib/mail/aliases**. If this data base does not exist, **smail** ignores it — its absence does not trigger an error condition. If **smail** encounters an error while it is resolving an address produced by an alias, it mails an error message to an address that has the string "owner-" prefixed to the name of the alias, if such a local address is defined.

Forward Files
A user may have a file named **.forward** in his home directory. If such a file exists, **smail** scans it for addresses. If a user has such a file in his home directory, **smail** directs all mail sent to that user to the address or addresses it contains. The file can contain addresses that specify other files or shell commands

as recipients.

If the **.forward** file is owned by **root** or by the user himself, then deliveries to any shell commands or files are performed under the user's user and group identifiers. If **smail** enters an error while it is resolving this list of addresses, it mails an error message to your system's postmaster.

In the **.forward** file for the user **root**, deliveries to shell commands and file addresses are performed under an unprivileged user and group identifier (by default, user **nobody**). The same is true for forward files that were not owned by **root** or by the given user. Finally, shell command and file addresses are not allowed at all in **.forward** files that are directories that can accessed by remote systems.

Mailbox Forwarding

As an alternate way to forward mail, the mailbox file for a user may contain a line of the form:

`Forward to` *address*, *address* ...

Onlyone line is read from this file, so addresses cannot be placed across multiple lines. The comments that apply to a **.forward** file also apply to this use of a mailbox file, except that **smail** assumes that a mailbox is not accessible by users on other systems.

A user is matched by name, either in upper or lower case, with delivery to that user being performed using a transport by the name of **local**. A user can also be matched by name if the user name is prefixed by **real-**. Delivery is performed by a transport named **local**.

Mailing Lists

Mailing list files can be created under a mailing-list directory — by default, directory **/usr/lib/mail/lists**. To create a new mailing list, create a file in this directory that contains a list of addresses. The basename of this file determines the local address that **smail** expands into this list of addresses. For example, a file named **info-smail** could be created, that contains a list of recipient addresses for a mailing list named "info-smail". **smail** then forwards any mail message mailed to address **info-smail** to every address in file **/usr/lib/mail/lists/info-smail**.

If **smail** encounters an error as it is attempting to deliver a mail message to an address within a list file, it mails an error message to a local address comprised of the base name of the list file prefixed with the string "owner-", if such an address is defined.

The Smart User

If **smail** cannot match a local address by any other means, it can forward that mail to another system — one that presumably has a more complete data base — via the director **smartuser**.

To declare another system to be a "smart user," set the attribute **smart_user** within file **/usr/lib/mail/config**. For example, attribute forwards mail to the host **mwc.com**:

```
smart_user = $user@mwc.com
```

If you do not set this attribute, then **smail** ignores the smart-user director.

Example Entries

The order of entries within **directors** determines the order in which operations are attempted. If a director matches an address, then **smail** calls no other director to expand or resolve that address. The following gives a version of **directors** that is equivalent to the default configuration:

```
# aliasinclude - expand ":include:filename" addresses
#    produced by alias files
aliasinclude:
      driver = aliasinclude,  # use this special-case driver
      nobody;                 # associate nobody user with addresses
#  when mild permission violations
#  are encountered

      copysecure,    # get permissions from alias director
      copyowners     # get owners from alias director

# forwardinclude - expand ":include:filename" addresses
#    produced by forward files
forwardinclude:
      driver = forwardinclude, # use this special-case driver
      nobody;
```

```
      copysecure,    # get perms from forwarding director
      copyowners     # get owners from forwarding director

# aliases - search for alias expansions stored in a database
aliases:
      driver = aliasfile,      # general-purpose aliasing director
      -nobody,                 # all addresses are associated
                               # with nobody by default, so setting
                               # this is not useful.
      owner = owner-$user;     # problems go to an owner address

      file = /usr/lib/aliases,
      modemask = 002,
      optional,                # ignore if file does not exist
      proto = lsearch

# dotforward - expand .forward files in user home directories
dotforward:
      driver = forwardfile,    # general-purpose forwarding director
      owner = Postmaster,      # problems go to the user's mailbox
      nobody,
      sender_okay;             # sender never removed from expansion

      file = ~/.forward,       # .forward file in home directories
      checkowner,              # the user can own this file
      owners = root,           # or root can own the file
      modemask = 002,          # it should not be globally writable
      caution = daemon:root,   # don't run things as root or daemon
                               # be extra careful of remotely
                                   # accessible home directories
      unsecure = "~ftp:~uucp:~nuucp:/tmp:/usr/tmp"

# forwardto - expand a "Forward to " in user mailbox files
#
# This emulates the V6/V7/System-V forwarding mechanism which uses a
# line of forward addresses stored at the beginning of user mailbox
# files prefixed with the string "Forward to "
forwardto:
      driver = forwardfile,
      owner = Postmaster, nobody, sender_okay;

      file = /usr/mail/${lc:user}, # the mailbox file for System V
      forwardto,                   # enable "Forward to " function
      checkowner,                  # the user can own this file
      owners = root,               # or root can own the file
      modemask = 0002,             # under System V, group mail can write
      caution = daemon:root        # don't run things as root or daemon

      # user - match users on the local host with delivery to their mailboxes
      user:     driver = user;# driver to match usernames

      transport = local        # local transport goes to mailboxes

# real_user - match usernames when prefixed with the string "real-"
#
# This is useful for allowing an address which explicitly delivers to
# a user's mailbox file.  For example, errors in a .forward file
# expansion can be delivered here, or forwarding loops between
# multiple machines can be resolved by using a real-username address.
real_user:
      driver = user;

      transport = local,
      prefix = "real-"         # for example, match real-root
```

```
# lists - expand mailing lists stored in a list directory
#
# mailing lists can be created simply by creating a file in the
# /usr/lib/smail/lists directory.
lists:    driver = forwardfile,
      caution,                # flag all addresses with caution
      nobody,                 # and then associate the nobody user
      owner = owner-$user;    # system V sites may wish to use
# o-$user, as owner-$user may be
# too long for a 14-char filename.

      # map the name of the mailing list to lower case
      file = lists/${lc:user}

# smart_user - a partially specified smartuser director
#
# If the config file attribute smart_user is defined as a string such
# as "$user@domain-gateway" then users not matched otherwise will be
# sent off to the host "domain-gateway".
#
# If the smart_user attribute is not defined, this director is ignored.
smart_user:
      driver = smartuser;     # special-case driver

# do not match addresses which cannot be made into valid
# RFC822 local addresses without the use of double quotes.
      well_formed_only
```

See Also

Administering COHERENT, config [smail], .forward, 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**.

directory — Definition

A **directory** is a table that maps names to files; in other words, it associates the names of a file with their locations on the mass storage device. Under some operating systems, directories are also files, and can be handled like a file.

Directories allow files to be organized on a mass storage device in a rational manner, by function or owner.

See Also

file, Using COHERENT
POSIX Standard, §5.1.2

dirent.h — Header File

Define directory-related data elements
#include <dirent.h>

dirent.h defines the data type **DIR** and the structure **dirent**. It is used with the portable directory-manipulation routines **closedir()**, **getdents()**, **opendir()**, **readdir()**, **rewinddir()**, and **telldir()**.

See Also

closedir(), getdents(), header files, opendir(), readdir(), rewinddir(), telldir()
POSIX Standard, §5.1.1

dirname — Command

Extract a directory name
dirname *string*

The command **dirname** extracts a directory name from a file's full path name. In effect, it is the complement of the command **basename**.

If *string* contains one or more slashes '/' plus text, then **dirname** prints out the portion of *string* up to (but not including) the last slash. For example, if *string* points to **/bin/sh**, then **dirname** will return **/bin**.

If *string* does not contain a slash or is empty (that is points to the current directory), **dirname** prints a single period '.'. For example, if *string* points to **myprogram**, then **dirname** returns a period.

Finally, if *string* consists only of one slash (that is, indicates the root directory), then **dirname** returns **/**.

See Also

basename, commands, cut, paste

dirs — Command

Print the contents of the directory stack
dirs

The COHERENT shell **sh** maintains an internal "directory stack", which is a stack of names of directories. You can manipulate this stack should you, for any reason, wish to traverse a number of directories quickly and efficiently.

The command **dirs** prints the current contents of the directory stack.

See Also

commands, popd, pushd, sh

disable — Command

Disable a port
/etc/disable *port...*

disable tells the COHERENT system not to create a login process for each given asynchronous *port*. For example, the command

```
/etc/disable com1r
```

disables port **/dev/com1r**. **disable** changes the entry for each given *port* in the terminal characteristics file **/etc/ttys**, and signals **init** to rescan the **ttys** file.

The command **enable** enables a port. The command **ttystat** checks whether a port is enabled or disabled.

Files

/etc/ttys — Terminal characteristics file

See Also

asy, commands, enable, login, ttys, ttystat

Diagnostics

disable normally returns one if it disables the *port* successfully and zero if not. If more than one *port* is specified, **disable** returns the success or failure status of the last port it finds. It returns -1 if it cannot find any given *port*. An exit status of -2 indicates an error.

div() — General Function (libc)

Perform integer division
#include <stdlib.h>
div_t div(*numerator***,** *denominator***)**
int *numerator***,** *denominator***;**

div() divides *numerator* by *denominator*. It returns a structure of the type **div_t**, which is structured as follows:

```
typedef struct {
      int quot;
      int rem;
} div_t;
```

div() writes the quotient into **quot** and the remainder into **rem**.

The sign of the quotient is positive if the signs of the arguments are the same; it is negative if the signs of the arguments differ. The sign of the remainder is the same as the sign of the numerator.

If the remainder is non-zero, the magnitude of the quotient is the largest integer less than the magnitude of the algebraic quotient. This is not guaranteed by the operators **/** and **%**, which merely do what the machine implements for divide.

See Also

ldiv(), libc, stdlib.h
ANSI Standard, §7.10.6.2

Notes

The ANSI Standard includes this function to permit a useful feature found in most versions of FORTRAN, where the sign of the remainder will be the same as the sign of the numerator. Also, on most machines, division produces a remainder. This allows a quotient and remainder to be returned from one machine-divide operation.

If the result of division cannot be represented (e.g., because *denominator* is set to zero), the behavior of **div()** is undefined. *Caveat utilitor.*

do — C Keyword

Introduce a loop

do is a C control statement that introduces a loop. Unlike **for** and **while** loops, the condition in a **do** loop is evaluated *after* the operation is performed. **do** always works in tandem with **while**; for example

```
do {
      puts("Next entry? ");
      fflush(stdout);
} while(getchar() != EOF);
```

prints a prompt on the screen and waits for the user to reply. The **do** loop is convenient in this instance because the prompt must appear at least once on the screen before the user replies.

See Also

break, C keywords, continue, while
ANSI Standard, §6.6.5.2

domain — System Administration

Set your system's mail domain
/etc/domain

The file **/etc/domain** sets the domain that the COHERENT mail system uses to create your fully qualified domain name. Your fully qualified domain name is created by appending the contents of **/etc/domain** to the contents of **/etc/uucpname**, with an intervening '.'. Unless you have a registered domain name, the contents of this file should be **UUCP**.

For information on registering in the United States catch-all domain **.us**, send mail to:

```
us-domain-request@venera.isi.edu
```

UUNET Communications Services of Falls Church, Virginia, will help you set up your own domain for a modest fee. Contact **info@uunet.uu.net** for more information; or telephone them at 703-876-5050.

See Also

Administering COHERENT, mail, paths, uucpname

dos — Command

Manipulate files on MS-DOS file systems
dos [-]dFflrtx[*flags***] [***device***] [***file ...***]**

The command **dos** allows the COHERENT user to manipulate an MS-DOS file system, which may be either a hard-disk partition or a floppy disk. It can build an empty MS-DOS file system, label it, list the files in it, transfer files between it and COHERENT, or delete files from it.

The given *device* must be a special file that specifies an MS-DOS file system, such as floppy-disk drive **/dev/fha0** or hard-disk partition **/dev/at0a**. The default *device* is **/dev/dos**, which the system administrator should link to the most commonly used device name.

dos converts between the differing file-name conventions of COHERENT and MS-DOS. An MS-DOS *file* argument may be specified in lower or upper case, using '/' as the path-name separator. When transferring files from MS-DOS to COHERENT, **dos** converts an MS-DOS file name to a COHERENT file name in lower case only. If the MS-DOS file name contains no extension, the COHERENT file name contains no '.'. When transferring files from COHERENT to MS-DOS, **dos** converts all alphabetic characters in a COHERENT file name to upper case; if a period '.' appears at the beginning or end of a file name, **dos** converts it to '_'. **dos** truncates the part of the file name before the last '.' to a maximum of eight characters and truncates the extension to a maximum of three characters.

The command line must specify exactly one of the following *functions*.

d Delete each *file* from the MS-DOS file system. This option also allows the user to delete empty directories.

F Create an empty MS-DOS file system on a formatted diskette. This option is analogous to the COHERENT command **/etc/mkfs**. The COHERENT commands **/etc/fdformat** and **/etc/mkfs** initialize a COHERENT diskette in two steps. The MS-DOS command **format** initializes an MS-DOS diskette by performing both the physical and logical formatting operations with one command. To initialize an MS-DOS diskette under COHERENT, use the command **/etc/fdformat -v** *devicename*, followed by the command **dos F** *devicename*. If *file* is named, **dos** copies it to the boot block of the file system. The **dos** command cannot build a file system on a hard-disk partition.

f Force removal of **readonly** files on the MS-DOS side.

l Label the MS-DOS file system. The command line must specify exactly one *file* argument, which gives the label.

r Replace each *file* on the MS-DOS file system with the COHERENT file of the same name. If a given *file* argument specifies a COHERENT directory, **dos** replaces its subdirectories recursively to the MS-DOS file system unless the **s** flag is used. If no *file* is specified, **dos** copies all files in the current directory to the MS-DOS file system.

t List the files on the MS-DOS file system. If no *file* argument is given, **dos** lists the entire MS-DOS file system; otherwise, it lists each *file*. If a *file* argument specifies an MS-DOS subdirectory, **dos** lists its contents. **dos** lists directories first in alphabetical order, then ordinary files in alphabetical order.

x Extract each *file* from the MS-DOS file system to a COHERENT file of the same name. If a given *file* argument specifies an MS-DOS subdirectory, **dos** extracts its contents recursively unless the **s** flag is used. If no *file* is given, **dos** extracts all files from the MS-DOS file system to the current COHERENT directory.

The following *flags* are available.

a Perform ASCII newline conversion on file transfer. When moving files from COHERENT to MS-DOS, this option converts each COHERENT newline character '\n' (ASCII **LF**) to an MS-DOS end-of-line (ASCII **CR** and **LF**); when moving files from MS-DOS to COHERENT, it does the opposite. By default, **dos** performs binary file transfer, without newline conversion.

k Keep the file modification time (mtime) on extract and replace operations. By default, **dos** gives extracted or replaced files the current time. With this option, **dos** gives the extracted or replaced file the same time as the original file.

n List files in order of creation (newest file last) rather than in alphabetical order. This option applies only to the table-of-contents function. **dos** always lists directories before files, with or without the **n** option.

p Perform a piped extract or replace (for use in pipelines). The command line must specify exactly one *file* argument. For extract, **dos** reads the given *file* and writes it to the standard output. For replace, **dos** reads the standard input and writes it to the given *file*.

s Suppress extraction or replacement of subdirectories. By default, **dos** extracts or replaces subdirectories recursively.

v Verbose option. Provide additional information about each function performed.

[**1-9**] A digit specifies a logical drive number on an extended MS-DOS partition. For example, **dos tv2 /dev/at0c** lists the directory of the second logical drive on extended MS-DOS partition **/dev/at0c**.

dos Commands

dos is an obsolete command. It has largely been superceded by the following family of COHERENT commands that manipulate 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 a directory to/from an MS-DOS file system
dosdel	Delete a file from an MS-DOS file system
dosdir	List contents of an MS-DOS directory
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

For details, see these commands' entries within the Lexicon.

Examples

The first example copies all files located in directories **sources** and **include**, as well as any subdirectories, from floppy drive **/dev/fva1** to correspondingly named subdirectories in the current COHERENT directory:

```
dos xavk /dev/fva1 sources include
```

Note that **fva1** is a high-density, 3.5-inch floppy disk in floppy-disk drive 1 (a.k.a., drive B:). The files will be copied with ASCII newline conversion and will retain the time and date that they had under MS-DOS.

The next example copies a file from an MS-DOS partition on your hard disk. Suppose that **C:** is the primary MS-DOS partition on your first hard drive. The following command copies file **C:\AUTOEXEC.BAT** to **/autoexec.bat** in your COHERENT root partition:

```
dos xa /dev/at0a /autoexec.bat
```

You will want to use the **a** switch any time you are transferring a text file.

Suppose that the second partition on your first hard drive (COHERENT device **/dev/at0b**) is an extended MS-DOS partition with two logical drives, **D:** and **E:**. To copy a COHERENT text file **/tmp/foo** to **D:\TMP\FOO**, use the command

```
dos ra1 /dev/at0b /tmp/foo
```

To copy non-text file **frotz** in the current COHERENT directory to MS-DOS file **E:\DBF\AX\FROTZ**, use the command

```
dos rp2 /dev/at0b dbf/ax/frotz < frotz
```

See Also

commands, fdformat, mkfs, MS-DOS

Notes

dos is an obsolete command. It has been retained for compatibility with earlier versions of COHERENT. We urge you to use the other members in the **dos** family of commands, which have a cleaner syntax and are much easier to use.

dos does not check for unusual characters in a COHERENT file name or for file names that differ from other file names only in case.

The **dos** family of commands now support large file systems, such as those created by MS-DOS versions 4.0 and 5.0.

The COHERENT system's **dos** family of commands do not understand compressed MS-DOS file systems created by programs such as **Stacker** or MS-DOS 6.0 **dblspace**. If you are running MS-DOS with file compression, you must copy files to an uncompressed file system (for example, to an uncompressed floppy disk or to the uncompressed host for a compressed file system) to make them accessible to the COHERENT **dos** commands.

doscat — Command

Concatenate a file on an MS-DOS file system
doscat *device*:**[**/*directory*/**]***file*

doscat concatenates *file* that is in *directory* on an MS-DOS file system. *device* names the floppy-disk or hard-disk device that holds the file system to be modified, e.g., **/dev/fha0**. You can also build a file of aliases so that you can access the drives as **a:**, **b:**, etc. For details, see the Lexicon entry for **doscp**, which explains how to set up defaults for the **dos** family of commands.

file can name either a single file, or can contain a wildcard character to name more than one file. For example, the command

```
doscat a:foo.c
```

concatenates file **foo.c** which is on the file system contained in device whose alias is **a:** (as defined in file **/etc/default/msdos**). Likewise, the command

```
doscat 'c:/dirname/*.txt'
```

concatenates all files with the suffix **.txt** in directory **dirname**, which, in turn, is on the file system contained in device whose alias is **c:** (as defined in file **/etc/default/msdos**). In this form of the command, **doscat** concatenates the files in the alphabetical order of their names. Note that the tail of the command must be enclosed within apostrophes, or the shell will expand the '*' before it is read by **doscat**.

Files

/etc/default/msdos — Setup file

See Also

cat, commands, dos

Notes

doscat does not understand compressed MS-DOS file systems created by programs such as **Stacker** or MS-DOS 6.0 **dblspace**. If you are running MS-DOS with file compression, you must copy files to an uncompressed file system (for example, to an uncompressed floppy disk or to the uncompressed host for a compressed file system) to make them accessible to the **doscat**.

doscp — Command

Copy files to/from an MS-DOS file system
doscp [-abkmrv] *src dest*

doscp copies files between MS-DOS and COHERENT file systems. The MS-DOS file system can reside either on a floppy disk, or on an MS-DOS partition of a hard disk.

src names the file being copied and the file system where it resides; *dest* names the file system and directory into which the file is copied. The operating system that owns the *src* file is implied by the name of the file system on which it resides. An MS-DOS file system must be named using the device that holds it, such as floppy-disk drive **/dev/fha0** or hard-disk partition **/dev/at0a**. You can also build a file of aliases so that you can access the drives as **a**, **b**, etc. For details, see the section entitled *Configuring the dos Commands*, below.

doscp converts a file's name from one operating system's conventions to the other's. An MS-DOS file argument may be specified in lower or upper case, using '/' as the path-name separator. When transferring files from MS-DOS to COHERENT, **doscp** converts an MS-DOS file name to a COHERENT file name in lower case only. If the MS-DOS file name contains no extension, the COHERENT file name contains no '.'. When transferring files from COHERENT to MS-DOS, **doscp** converts all alphabetic characters in a COHERENT file name to upper case; if a period '.' appears at the beginning or end of a file name, **doscp** converts it to '_'. **doscp** truncates the portion of the file name to the left of the '.' to a maximum of eight characters and portion to the right of the '.' to a maximum of three characters.

doscp recognizes the following options:

a Perform ASCII newline conversion on file transfer. When moving files from COHERENT to MS-DOS, this option converts each COHERENT newline character '\n' (ASCII LF) to an MS-DOS end-of-line (ASCII CR and LF). When moving files from MS-DOS to COHERENT, it does the opposite. By default, **doscp** performs ASCII conversion on files that have an ASCII extention. See **Setup**, below.

b Do not perform any newline conversion on file transfers.

k Keep: give the copied file the same time stamp as its original. By default, **doscp** gives copied files the current time.

m Same as **a**, described above

r Same as **b**, described above.

v Verbose. Provide additional information about each action performed.

Configuring the dos Commands

The **dos** family commands read the file **/etc/default/msdos** before they begin to interpret arguments. By modifying this file, you can establish defaults that let COHERENT's **dos** commands resemble their counterparts under MS-DOS. You can set up two classes of defaults: *device* defaults and *file* defaults.

A device default lets you declare an alias for a device that holds an MS-DOS file system. This device can be a floppy-disk drive, a partition on a hard disk, or an extended partition on a hard disk. The alias must consist of one or two letters. No letter can serve as an alias for more than one device. For example, the following declaration:

```
c=/dev/at0a
```

specifies that the hard-disk partition accessed via device **/dev/at0a** is a "Primary MS-DOS" partition, and that its alias is **c**. Hereafter, the **dos** commands will interpret **c** as being equivalent to **/dev/at0a**.

The declaration

```
d=/dev/at0b;1
```

specifies the first "Extended MS-DOS" partition on the partition accessed via device **/dev/at0b**. Bumping the number from 1 to 2 would specify the second extended MS-DOS partition within partition **/dev/at0b**, as in:

```
e=/dev/at0b;2
```

Notice how the device names (c, d, and e) can correspond to the same drive names as under MS-DOS, whether or not they are primary or extended partitions.

File declarations, on the other hand, simply declare that all files with a given suffix are text files and should always have their newline characters converted from COHERENT to MS-DOS format (or vice versa). For example, placing the line

```
.c
```

in **/etc/default/msdos** tells all of the **dos** commands that all files with the suffix **.c** are text files and should have their newline characters converted by default. You can have any number of file defaults in **/etc/default/msdos**.

Examples

The first example copies all C source files from floppy drive **/dev/fva1** to correspondingly named files in the current COHERENT directory, preserves the time stamp, and performs newline conversion upon them:

```
doscp -akv /dev/fva1:source/\*.c .
```

Note that you must quote wildcard characters with a backslash to keep the shell from interpreting them. Also note that **/dev/fva1** is a high-density, 3.5-inch floppy disk in floppy-disk drive 1. So, if your **default** file contained the entry

```
b=/dev/fva1
.c
```

you could also have typed:

```
doscp -kv b:source/\*.c .
```

The next example copies a file from an MS-DOS partition on your hard disk to a COHERENT file system. Suppose that C is the primary MS-DOS partition on your first hard drive. The following command copies file **C:\AUTOEXEC.BAT** to **/tmp/autoexec.bat** in your COHERENT partition:

```
doscp /dev/at0a:autoexec.bat /tmp
```

If your **/etc/default** file contains the entry

```
c=/dev/at0a
```

then you can also type:

```
doscp c:autoexec.bat /tmp
```

Files

/etc/default/msdos — Setup file

See Also

commands, cp, dos

Notes

For a discussion of the error message

```
Probably not a DOS disk
```

see the notes to the Lexicon entry for **doscp**. **doscp** does not check for unusual characters in a COHERENT file name or for file names that differ from other file names only in case.

Beware of using **doscp** to create impossible files, e.g., **com1**. Such files create serious problems; for example, if you try to **TYPE** or otherwise perform MS-DOS operations on **com1**, you will attack the MS-DOS device driver instead of the file. Be sure to rename all such files when you copy them from a COHERENT to an MS-DOS file system.

doscp does not understand compressed MS-DOS file systems created by programs such as **Stacker** or MS-DOS 6.0 **dblspace**. If you are running MS-DOS with file compression, you must copy files to an uncompressed file system (for example, to an uncompressed floppy disk or to the uncompressed host for a compressed file system) to make them accessible to the **doscp**.

doscpdir — Command

Copy a directory to/from an MS-DOS file system
doscpdir [-akmv] *src dest*

doscpdir copies a directory and its contents between an MS-DOS file system and a COHERENT file system. The MS-DOS file system can reside either on a floppy disk, or on the MS-DOS segment of a hard disk on your system.

src names the directory being copied and the file system where it resides; *dest* names the file system and directory into which the file is copied. The operating system that owns the *src* file is implied by the name of the file system on which it resides. An MS-DOS file system must be named using the device that holds it, such as floppy-disk drive **/dev/fha0** or hard-disk partition **/dev/at0a**. You can also build a file of aliases so that you can access the drives as **a**, **b**, etc. For details, see the Lexicon entry for **doscp**, which explains how to set up defaults for the **dos** family of commands.

doscpdir converts a file's name from one operating system's conventions to the other's. An MS-DOS file argument may be specified in lower or upper case, using '/' as the path-name separator. When transferring files from MS-DOS to COHERENT, **doscpdir** converts an MS-DOS file name to a COHERENT file name in lower case only. If the MS-DOS file name contains no extension, the COHERENT file name contains no '.'. When transferring files from COHERENT to MS-DOS, **doscpdir** converts all alphabetic characters in a COHERENT file name to upper case; if a period '.' appears at the beginning or end of a file name, **doscpdir** converts it to '_'. **doscpdir** truncates the part of the file name before the last '.' to a maximum of eight characters and truncates the extension to a maximum of three characters.

doscpdir recognizes the following options:

a Perform ASCII newline conversion on file transfer. When moving files from COHERENT to MS-DOS, this option converts each COHERENT newline character '\n' (ASCII LF) to an MS-DOS end-of-line (ASCII CR and LF). When moving files from MS-DOS to COHERENT, it does the opposite. By default, **doscpdir** performs ASCII conversion on files that have an ASCII extention.

k Keep: give the copied file the same time stamp as its original. By default, **doscpdir** gives copied files the current time.

m Same as **a**, described above

v Verbose. Provide additional information about each action performed.

Example

The following command copies COHERENT directory **/usr/src** to directory **/mydir** on the MS-DOS file system. It assumes that you have set **c** as a default for a hard-disk device:

```
doscpdir -va /usr/src c:/mydir
```

Files

/etc/default/msdos — Setup file

See Also

commands, cpdir, dos

Notes

doscpdir does not check for unusual characters in a COHERENT file name or for file names that differ from other file names only in case.

doscpdir does not understand compressed MS-DOS file systems created by programs such as **Stacker** or MS-DOS 6.0 **dblspace**. If you are running MS-DOS with file compression, you must copy files to an uncompressed file system (for example, to an uncompressed floppy disk or to the uncompressed host for a compressed file system) to make them accessible to the **doscpdir**.

dosdel — Command

Delete a file from an MS-DOS file system
dosdel [-fv] *device*:**/***dir***/***file*

dosdel deletes *file* that lives on MS-DOS file-system *device*. The MS-DOS file system can reside either on a floppy disk, or on the MS-DOS segment of a hard disk on your system. The MS-DOS file system must be named using the device that holds it, such as floppy-disk drive **/dev/fha0** or hard-disk partition **/dev/at0a**. You can also build a file of aliases so that you can access the drives as **a**, **b**, etc. For details, see the Lexicon entry for **doscp**, which explains how to set up defaults for the **dos** family of commands.

dosdel takes the following options:

f Force removal of **readonly** files.

v Verbose output: provide additional information about each action.

Example

The following command deletes **myfile**. It assumes that you have defined **c** to be a default for a device upon which you have set an MS-DOS file system:

```
dosdel c:/mydir/myfile
```

Files

/etc/default/msdos — Setup file

See Also

commands, dos

Notes

dosdel does not understand compressed MS-DOS file systems created by programs such as **Stacker** or MS-DOS 6.0 **dblspace**. If you are running MS-DOS with file compression, you must copy files to an uncompressed file system (for example, to an uncompressed floppy disk or to the uncompressed host for a compressed file system) to make them accessible to the **dosdel**.

dosdir — Command

List contents of an MS-DOS directory
dosdir [-nv] *device*:**[***dir***/][***file***]**

dosdir lists the contents of a *directory* that lives on an MS-DOS file system. The MS-DOS file system can reside either on a floppy disk, or on the MS-DOS segment of a hard disk on your system. The MS-DOS file system must be named using the device that holds it, such as floppy-disk drive **/dev/fha0** or hard-disk partition **/dev/at0a**. You can also build a file of aliases so that you can access the drives as **a**, **b**, etc. For details, see the Lexicon entry for **doscp**, which explains how to set up defaults for the **dos** family of commands.

dosdir recognizes the following options:

n Newest: List the files in the order in which they were last modified, from newest to oldest. By default, **dosdir** lists files in alphabetical order.

v Verbose. Provide additional information about each action performed.

Example

The following command lists the contents of **mydir**. It assumes that you have defined **c** as a default for a device on which is set an MS-DOS file system:

```
dosdir c:/mydir
```

Files

/etc/default/msdos — Setup file

See Also

commands, **dos**, **dosls**, **ls**

Notes

If you see the error

```
dosdir: Probably not a DOS disk (media descriptor 0x00)
```

dosdir cannot find a valid boot block on a partition. It happens when you try to access an extended DOS partition as though it were a primary partition. Check the line in **/etc/default/msdos** to see how **dosdir** is accessing that partition.

For example, if are trying to access device **h:** and the entry for that device reads

```
h=/dev/sd1a
```

this device may in fact be an extended partition. It sometimes happens with removable media, such as removable SCSI disks, have extended partitions built on them without the operator's knowledge. To test whether this partition is in fact an extended partition, type the command:

```
dosdir -v /dev/sd1a;1
```

If you then see the contents of the partition, you know that you are on the right track. Change the entry for device **h** to read

```
h=/dev/sd1a;1
```

and all should be well.

dosdir does not understand compressed MS-DOS file systems created by programs such as **Stacker** or MS-DOS 6.0 **dblspace**. If you are running MS-DOS with file compression, you must copy files to an uncompressed file system (for example, to an uncompressed floppy disk or to the uncompressed host for a compressed file system) to make them accessible to the **dosdir**.

dosformat — Command

Build an MS-DOS file system
dosformat [-v] *device***:**

dosformat builds an MS-DOS file system on a floppy disk. The disk must first have been formatted with the command **fdformat -v**. *device* names the floppy-disk drive that holds the disk to receive the file system, such as **/dev/fha0**. See the Lexicon entry **floppy disks** for a table of the COHERENT floppy-disk devices. You can also build a file of aliases so that you can access the drives as **A**, **B**, etc. For details, see the Lexicon entry for **doscp**, which explains how to set up defaults for the **dos** family of commands. Note that the device name must always be suffixed with a colon ':', just like an MS-DOS device name.

The option **-v**, tells **dosformat** to provide additional information about each action it performs.

Example

The following example formats a disk. It assumes that you have defined **a** as a default for a device upon which is set an MS-DOS file system:

```
dosformat a:
```

Files

/etc/default/msdos — Setup file

See Also

commands, dos, fdformat

Notes

To create a double-sided, double-density formatted floppy disk in drive 0 (drive A), use **/dev/fqa0** for 3.5-inch disks, or **/dev/f9a0** for 5.25-inch disks.

doslabel — Command

Label an MS-DOS floppy disk
doslabel [-v] *device***:** *label*

doslabel puts *label* onto an MS-DOS floppy disk. *device* names the floppy-disk drive that holds the disk to be labelled, such as **/dev/fha0**. See the Lexicon entry **floppy disks** for a table of the COHERENT floppy-disk devices. You can also build a file of aliases so that you can access the drives as **a**, **b**, etc. For details, see the Lexicon entry for **doscp**, which explains how to set up defaults for the **dos** family of commands.

The option **-v**, tells **doslabel** to provide additional information about each action it performs.

Example

The following command labels an MS-DOS floppy disk with the string **mydisk**. It assumes that you have defined **a** as a default for a device that holds an MS-DOS file system:

```
doslabel a: mydisk
```

Files

/etc/default/msdos — Setup file

See Also

commands, dos

dosls — Command

List files on an MS-DOS file system
dosls [-v] *device***:[/***directory***/][***file***]**

dosls lists all files in *directory* on an MS-DOS file system. *device* names the floppy-disk or hard-disk device that holds the file system to be modified, e.g., **/dev/fha0**. You can also build a file of aliases so that you can access the drives as **a**, **b**, etc. For details, see the Lexicon entry for **doscp**, which explains how to set up defaults for the **dos** family of commands.

The option **-v** tells **dosls** to print its output in a long format, analogous to what the command **ls -l** prints.

Example

The following displays the contents of directory **src**. It assumes that you have defined **c** as a default for a device on which you have set an MS-DOS file system:

```
dosls -v c:/src
```

Files

/etc/default/msdos — Setup file

See Also

commands, dos, dosdir, ls

Notes

dosls does not understand compressed MS-DOS file systems created by programs such as **Stacker** or MS-DOS 6.0 **dblspace**.

dosmkdir — Command

Create a directory in an MS-DOS file system
dosmkdir *device***:***directory*

dosmkdir makes *directory* in an MS-DOS file system. *device* names the floppy-disk or hard-disk device that holds the file system to be modified, e.g., **/dev/fha0**. You can also build a file of aliases so that you can access the drives as **a**, **b**, etc. For details, see the Lexicon entry for **doscp**, which explains how to set up defaults for the **dos** family of commands.

Example

The following command creates directory **mydir**. It assumes that you have defined **a** to be a device in which is set an MS-DOS file system:

```
dosmkdir a:/mydir
```

Files

/etc/default/msdos — Setup file

See Also

commands, dos, mkdir

Notes

dosmkdir does not understand compressed MS-DOS file systems created by programs such as **Stacker** or MS-DOS 6.0 **dblspace**.

dosrm — Command

Remove a file from an MS-DOS file system
dosrm *device***:[/***directory***/]***file*

dosrm removes *file* from *directory* on an MS-DOS file system. *device* names the floppy-disk or hard-disk device that holds the file system to be modified, e.g., **/dev/fha0**. You can also build a file of aliases so that you can access the drives as **a**, **b**, etc. For details, see the Lexicon entry for **doscp**, which explains how to set up defaults for the **dos** family of commands.

Example

The following deletes all **.c** files on an MS-DOS disk. It assumes that you have defined **b** to be a device on which you have set an MS-DOS file system:

```
dosrm 'b:*.c'
```

Files

/etc/default/msdos — Setup file

See Also

commands, dos, dosrmdir, rm

Notes

dosrm does not understand compressed MS-DOS file systems created by programs such as **Stacker** or MS-DOS 6.0 **dblspace**.

dosrmdir — Command

Remove a directory from an MS-DOS file system
dosrmdir *device***:***directory*

dosrmdir removes *directory* from an MS-DOS file system. *device* names the floppy-disk or hard-disk device that holds the file system to be modified, e.g., **/dev/fha0**. You can also build a file of aliases so that you can access the drives as **a**, **b**, etc. For details, see the Lexicon entry for **doscp**, which explains how to set up defaults for the **dos** family of commands.

Example

The following command removes directory **foo**. It assumes that you have defined **a** to be a device in which you have set a disk with an MS-DOS file system:

```
dosrmdir c:/foo
```

Files

/etc/default/msdos — Setup file

See Also

commands, dos, dosrm, rmdir

Notes

dosrmdir does not understand compressed MS-DOS file systems created by programs such as **Stacker** or MS-DOS 6.0 **dblspace**.

double — C Keyword

Data type

A **double** is the data type that encodes a double-precision floating-point number. On most machines, **sizeof(double)** is defined as four machine words, or eight **char**s. If you wish your code to be portable, do *not* use routines that depend on a **double** being 64 bits long. The ranges of values that can be held by a COHERENT **double** are set in header file **float.h**.

Different formats are used to encode **double**s on various machines. These formats include IEEE, DECVAX, and BCD (binary coded decimal), as described in the entry for **float**. COHERENT 286 uses DECVAX format; COHERENT 386 uses IEEE format.

See Also

C keywords, data formats, float, float.h, portability
ANSI Standard, §6.1.2.5

dpac — Command

De-fragment a COHERENT file system
dpac [-q] *raw_device*

Command **dpac** de-fragments the COHERENT file system on *raw_device*. Defragmentation leaves each file in the file system physically contiguous. This reduces the number of seeks needed to access a file, and threfore permits disk I/O to run at its maximum speed. The default algorithm also sorts the i-nodes by modification date and puts the oldest ones at the beginning of the partition. This helps the file system remain un-fragmented longer.

You must **umount** the target file system *raw_device* before you run **dpac** on it. Failure to do so will corrupt the file system. For example, the command

```
dpac /dev/rat0a
```

tells **dpac** to map the first partition on the first drive and prompt whether to continue. *raw_device* must be a partition or a floppy disk rather than an entire hard drive.

dpac begins by making a map of the file system. It displays a histogram of its activity as it builds the map; this lets you see what the kernel must do in order to access each file. When it has finished the file system map, **dpac** prompts you and asks whether to quit, continue with defragmentatation using the default date sort, or to continue but to use an unsorted method of defragmentation. **dpac** does not use terminfo or termcap for its display, and is intended for use on the console's **ansipc** terminal setting. This lets you run it from a bootable floppy disk.

See Also

commands, fmap, fsck, qpac, spac, upac

Notes

To see how fragmented a file system is, use the command **fmap**.

Note that you can also de-fragment a file system by copying it to a tape, then deleting it and restoring it from the tape. Another method of defragmentation is to use the command **cpdir** to copy the file system to a spare partition (should you have one that is large enough), then using the spare partition in place of the old partition.

Please note that if you use **dpac** incorrectly or without sufficient amounts of RAM or spare disk space, you can damage or destroy your file system. *Never* run **dpac** on the partition-table device (e.g., /dev/at0x), or on the root device. *Caveat utilitor!*

dpac was written by Randy Wright (rw@rwsys.wimsey.bc.ca).

drand48() — Random-Number Function (libc)

Return a 48-bit pseudo-random number as a double
double drand48()

Function **drand48()** generates and returns a 48-bit pseudo-random number in the form of a **double**.

See Also

libc, srand48()

drvld.all — System Administration

Load loadable drivers at boot time
/etc/drvld.all

The file **/etc/drvld.all** holds commands to load loadable drivers into memory when you boot the COHERENT system. It is read from the script **/etc/brc**, which is executed whenever the COHERENT system is rebooted into single-user mode.

Under COHERENT 286, **drvld.all** (as its name implies) includes calls to the command **drvld** to load loadable drivers. COHERENT 386 does not implement loadable device drivers; however, it uses **drvld.all** to load the keyboard table and perform other useful work.

See Also

Administering COHERENT, brc, keyboard

du — Command

Summarize disk usage
du [-a] [-s] [*directory ...***]**

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

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

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

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

See Also

commands, df, find

Notes

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

dump — Command

File-system backup utility
dump [*options***] [***argument ...***]**

dump dumps either all or a portion of file system *argument* to magnetic tape or floppy disks. File-system dumps are in a format that permits you to restore all or some of the files to the original file system, and to select files either by name or by i-number.

A file-system dump includes all files changed since the *dump since* date, plus each file's full path name (for the benefit of **dumpdir**).

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

0-9 The digit gives the level number of the dump. The dump-since date is the most recent date in the dump-date file **/etc/ddate** that is (1) associated with this file system and (2) has a level number less than the current dump level. For example, if you request a level-3 dump, **dump** will back up all files not backed up since the last level-2 dump. A level-0 dump by definition backs up all files in the file system.

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

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

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

s The next *argument* gives the length of the dump tape in feet. **dump** keeps a running total of the quantity of tape it has written, and it asks for a new reel if it appears that the end of the reel is near. The default length is 2,300 feet.

S The next *argument* gives the size of the dump output device, in blocks. This is used only if you are backing up the file system to floppy disks or streaming cartridge tape rather than to nine-track magnetic tape.

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

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

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

Files

/dev/dump — Default dump device
/etc/ddate — Dump date file

See Also

badscan, commands, dumpdate, dumpdir, restor

Diagnostics

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

dump requires that its output be written to disks that are free of bad sectors. If you write a dump to a disk with bad sectors, you will not be able to restore files from that disk.

When formatting disks to be used with **dump**, use the command

```
/etc/fdformat -v device
```

This forces **fdformat** to verify the format. It takes twice as long, but it ensures that the disk is good at least at a first level of testing. Reject any disks that have any defects — or save them for use with COHERENT file systems, which can map out bad sectors.

Notes

Please note that **dump** is now regarded as being obsolete. We strongly encourage users to use **cpio** instead.

dumpdate — Command

Print dump dates
dumpdate [*filesystem ...***]**

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

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

Files

/etc/ddate — Dump date file

See Also

commands, dump, dumpdir, restor

dumpdir — Command

Print the directory of a dump
dumpdir [af [*argument ...***]]**

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

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

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

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

Files

/dev/dump — Default dump device
/tmp/dd*XXXXXX* — To hold directory blocks

See Also

commands, dump

Diagnostics

The dump/restore format puts a header at the beginning of the dump that includes all the information about what lives where in the dump. **dumpdir** reads this header to discover what files are in the dump. If the header is too large to fit onto one disk, **dumpdir** will then prompt you to insert the additional disk or disks; if this happens, insert the requested disk and then type **<return>**.

Notes

dump requires that its output be written to disks that are free of bad sectors. If you write a dump to a disk with bad sectors, you will not be able to restore files from that disk. For details on using disks with **dump**, see its Lexicon entry.

dumptape.h — Header File

Define data structures used on dump tapes
#include <dumptape.h>

dumptape.h defines the data structures used on archives dumped with the command **dump**. Note that the command **dump** is regarded as obsolete. In its place, you should use **pax**, **tar**, or **cpio**.

See Also

header files

Notes

This header file is obsolete, and will be dropped from a future release of COHERENT. Its use is strongly discouraged.

dup() — System Call (libc)

Duplicate a file descriptor
#include <unistd.h>
int dup(*fd***) int** *fd***;**

dup() duplicates the existing file descriptor *fd*, and returns the new descriptor. The returned value is the smallest file descriptor that is not already in use by the calling process.

See Also

dup2(), fopen(), fdopen(), libc, stdio.h, unistd.h
POSIX Standard, §6.2.1

Diagnostics

dup() returns a number less than zero when an error occurs, such as a bad file descriptor or no file descriptor available.

dup2() — General Function (libc)

Duplicate a file descriptor
#include <unistd.h>
int dup2(*fd*, *newfd***) int** *fd*, *newfd***;**

dup2() duplicates the file descriptor *fd*. Unlike its cousin **dup()**, **dup2()** allows you to specify a new file descriptor *newfd*, rather than having the system select one. If *newfd* is already open, the system closes it before assigning it to the new file. **dup2()** returns the duplicate descriptor.

See Also

dup(), libc, stdio.h, unistd.h
POSIX Standard, §6.2.1

Diagnostics

dup2() returns a number less than zero when an error occurs, such as a bad file descriptor or no file descriptor available.