Using the COHERENT System

This tutorial introduces the COHERENT system. It introduces such basic concepts as *command* and *file system,* and walks you through simple exercises to help you gain some familiarity with the dimensions of COHERENT. If you are new to COHERENT, you should read through this tutorial first. Not every section in here will be immediately useful to every user; for example, a beginner will probably not need to study the section on system administration, at least at first. But sooner or later, you will need to work with all of the material in this tutorial.

If you are unfamiliar with what an *operating system* is, or if you are unsure how COHERENT differs from other operating systems (such as MS-DOS), turn to the Lexicon article for COHERENT. There, you will find a brief description of what an operating system is and what makes COHERENT special.

Before you can begin to use this tutorial, you must install COHERENT on your computer. If you have not yet done so, turn to the Release Notes that came with this manual and follow the directions in them.

How Do I Begin?

For everyone, there's that first time. You have installed COHERENT on your computer, you've checked the file system, mounted all of your file systems, and have gone into multi-user mode. Now you are sitting in front of your computer and all you see on your screen is the enigmatic phrase:

```
Coherent 386 login:
```

"What," you ask yourself, "do I do now?" Well, the rest of this section will tell you how to get started with COHERENT.

Logging in

To begin, you must *log in*. Unlike MS-DOS, COHERENT is a multi-user system: many people can use the same computer. They can access it either via terminals that you plug into your computer's serial ports, or via modem. Each user owns his personal set of files, his special way of setting up his environment, his own mailbox, and other things which are special to him alone. Because many people can use COHERENT, before you begin to work with COHERENT you must tell it who you are. This process of identifying yourself to COHERENT is called *logging in*. That mysterious prompt

```
Coherent 386 login:
```

is COHERENT's way of asking you who you are.

To log in, type your personal login identifier. You set this identifier when you installed COHERENT onto your computer. Most people set their login identifier to their initials or their first names, usually all in lower-case letters. Once you type your login identifier, press the `(¢)` key (sometimes labelled **<Enter>** or **<Return>**). If you did not set up a login for yourself during installation, log in as the superuser **root** and add one for yourself. For information on how to log in as the superuser, see below. For information on how to add a new user, see the section on **Adding a New User**, below, or see the Lexicon article for the command **newusr**.

While you were installing COHERENT on your system, you were given the option of setting a password for your login identifier. This is done to stop other users from logging in as you — and to keep "crackers" from dialing into your system and vandalizing it. If you did set a password, after you enter your login identifier COHERENT prompts you for it with the following prompt:

```
Password:
```

Type your password. Note that COHERENT does *not* display the password on the screen as you type it; this is to prevent bystanders from seeing your password over your shoulder as you type it. After you type your password, again type `(¢)`.

If you entered your login identifier and passwords correctly, COHERENT will display the command prompt:

```
$
```

This is COHERENT's way of saying, "Give me a command, I'm ready to go!" If you made a mistake while logging in, either with your login identifier or your password, COHERENT will reply,

```
Login incorrect:  try again.
```

and again display its login prompt:

```
Coherent 386 login:
```

Try again, until you do manage to log in. If you have received the ‘$’ prompt, congratulations! COHERENT is now ready to work with you.

Special Terminal Keys

The next sections will introduce you to some elementary COHERENT commands. Before we continue, however, you must first become familiar with a few special keys on your computer’s keyboard, and with the special meanings they have to the COHERENT system.

One special key on the keyboard will be used frequently in your work: the (¢) key. As noted above, this key is sometimes labelled **<Enter>**.

You must conclude every command you type into COHERENT by pressing (¢). This tells COHERENT that you have finished typing, and that you now want it to execute your command. COHERENT will not execute your command until you press this key.

Another special key is the **control** key. This key is usually labelled **Ctrl** or **cntl** or **cont**. Most terminals place it to the left of the keyboard. This key is used to send certain special characters.

The **ctrl** key is like another kind of shift key: to use it, hold it down while you press another key. For example, to send the computer a **<ctrl-D>** character, hold down the **ctrl** key, strike the **D** key, then release both keys.

Because control characters have no corresponding printable characters, in this tutorial they will be represented in the form:

```
<ctrl-D>
```

for the character **ctrl-D**.

While you are typing information into the COHERENT system, you can correct what you type before COHERENT processes it. Two keys will help you do this. The first is the **<kill>** character, which erases the line entirely and allows you to begin again. This is usually **<ctrl-U>**.

The other key is the **<erase>** character, normally **<ctrl-H>** or the **<backspace>** key. This moves the cursor one character to the left, to erase the most recently typed character.

One more special key is the **<interrupt>** key. This key aborts a command before it normally finishes. By default, **<ctrl-C>** is the abort key on your keyboard.

Try Some COHERENT Commands

Now that you’ve logged in to your COHERENT system, try a few simple COHERENT commands to get a feel for COHERENT. Type the following examples just as they are shown, and observe what COHERENT does in response to each. Be sure to press (¢) to end each line.

The first example uses the command **cat**, to let you type a small chunk of text and save it in a file.

```
cat >file01
This is a sample COHERENT file.
<ctrl-D>
```

Remember, don’t type **<ctrl-D>** literally — rather, hold down the **ctrl** key and press ‘D’ at the same time.

In the above script, the characters **cat** tell COHERENT to invoke its concatenation program. The characters **>file01** tells COHERENT to write what you type into a file that you name **file01**. The line

```
This is a sample COHERENT file.
```

is the text that COHERENT writes into **file01**. Finally, **<ctrl-D>** signals COHERENT that you have finished typing.

Now type:

```
cat file01
```

This command again invokes the concatenation program **cat**, but this time tell it to print on your screen the contents of **file01**, which you just created. In reply to your command, COHERENT should print on your screen:

```
This is a sample COHERENT file.
```

which is the text you typed in the previous exercise.

Finally, type the command:

```
lc
```

This command lists all of the files that you have in the current directory. In reply to your command, COHERENT should print on your screen:

```
Files:
    file01
```

which is the file you just created. (You may see other files as well.)

Congratulations! You have just made COHERENT work for you.

To review: The first command, **cat**, created a file and filled it with some text. The second **cat** command copied the file onto your terminal's screen. Finally, the command **lc** printed the name of each of your files. The following sections of this tutorial describe each of these commands in more depth. Each command also has its own entry in the Lexicon, which appears in the second half of this manual; look there for a full description of each command, what it does, and how you can use it.

Giving Commands to COHERENT

Once you have logged into COHERENT, all of its resources are yours to command. COHERENT's *commands* give you control over these resources.

Every COHERENT command has the same structure: the *command name*, which tells COHERENT the command you want it to execute; and the *arguments*, which detail what you want the command to do, how you want it to do it, and to what you want it done.

Some commands consist only of the command name, and do not take arguments. For example, the command

```
lc
```

which was introduced in the previous section, has **lc** as the first part and prints the names of all files in the current directory, in columns. If you have no files, **lc** prints nothing.

The second part of the command consists of the *arguments* given to the command. (These are also known by the term *parameters*.) Arguments are separated from each other by spaces or tab characters.

The arguments of the command are further divided into *options* and *names*. *Names* usually name files; *options* modify the action of the command. An option is usually prefixed by a hyphen '-'.

An example of a *name* argument is shown in this example of a **cat** command:

```
cat file01
```

This command types the contents of **file01** on your terminal. The name argument is **file01**.

For an example of options, consider the command **ls**. **ls** lists your file names one name per line. Thus, typing

```
ls
```

produces a list of the form:

```
file01
```

However, **ls** can tell you more about a file than just its name. To see additional information about each file, type:

```
ls -l
```

The '-l' option to **ls** prints a "long" output, of the following form:

```
-rw-r--r-- 1 you    17 Sat Aug 15 17:20 file01
```

This listing shows the size of the file, the date it was created or last modified, and its degree of protection. The letters to the left of the listing give the permissions for the file; these describe who is allowed to do what to the file. These are described in detail in the Lexicon articles for the commands **ls** and **chmod**. The other entries on that line respectively name the owner of the file (in this case, *you*); the size of the file, in bytes; the date and time the file was last modified; and, finally, the file's name.

As an example of combining an option parameter with a name parameter, consider the command:

```
ls -l file01
```

This invokes the command **ls**, tells it to print a long listing, and tells it to list only the file **file01**.

As you will see in the following sections, almost all COHERENT commands have this syntax.

help, man, apropos: Help with Commands

The COHERENT system has three commands that give information about other commands: **help**, which prints a brief summary of how to use a command; **man**, which prints the full Lexicon entry for that command on your screen; and **apropos**, which shows all commands (all Lexicon entries, really) which relate to a given subject.

To find out about the **help** command, type

```
help
```

by itself, or type:

```
help help
```

The latter command tells **help** to print the help entry for the **help** command itself.

To get information on the **lc** command, type:

```
help lc
```

You will see something very like the following:

```
lc -- List directory's contents in columnar format

lc [ -1abcdfp ] [ directory ...]

Options:
     -1        List files one per line instead of in columns
     -a        List all files in directory (including '.' and '..')
     -b        List block-special files only
     -c        List character-special files only
     -d        List directories only
     -f        List regular files only
     -p        List pipe files only

Options can be combined.  If no directory is specified, the current
directory is used.
```

To obtain detailed information on a command, use the **man** command. (**man** is short for “manual”.) As noted above, the **man** prints on your screen a duplicate of that command’s entry in the Lexicon. To learn more about the **help** command, type:

```
man help
```

If your screen fills with information, **man** will wait for you to press the spacebar to continue. This is to prevent you from missing information should it scroll too fast.

Finally, the command **apropos** print information about all Lexicon articles that are *a propos* a given topic. For example, if you want to know what Lexicon articles are *a propos* the subject of printers, type the command:

```
apropos printer
```

COHERENT replies by printing something like the following:

```
chreq   Change priority, lifetime, or printer for a job
epson   Prepare files for Epson printer
hp      Prepare files for Hewlett-Packard LaserJet printer
hpd     Spooler daemon for laser printer
lp      Spool a job for printing
lpd     Spooler daemon for line printer
lpioctl.h       Definitions for line-printer I/O control
lpr     Spool a job for printing on a dot-matrix printer
lpshut  Turn off the printer daemon despooler
lpskip  Abort/restart current job on line printer
lpstat  Give status of printer or job
printer How to attach and run a printer
prps    Prepare files for PostScript-compatible printer
route   Show or reset a user's default printer
```

Read the summary descriptions of each Lexicon article to see which ones look promising; then either look them up in this manual, or use the **man** command to display them on your screen.

Our survey of elementary commands will conclude by describing two important tasks: how to reboot the computer, and how to log out.

Shutting Down COHERENT and Rebooting

Under many operating systems, such as MS-DOS, rebooting is as simple as pressing a couple of keys or cycling power on the computer. The COHERENT system, however, is a multi-user, multi-tasking operating system that is more sophisticated than MS-DOS or similar operating systems. COHERENT maintains an elaborate system of internal buffers that are designed to reduce the frequency with which a program has to read data from, or write data to, the hard disk. If you were just to turn the computer off and turn it on again, all of the data in those buffers would be lost. At the very least, each user would lose whatever data he was working with at the time; at worst, the COHERENT file system could be damaged and files lost.

For this reason, it is extremely important that you shut down COHERENT properly. You *must* follow these procedures if you want to shut off the computer, or if you wish to reboot MS-DOS.

To shut down COHERENT, do the following:

- When you see the COHERENT command prompt, type either **<ctrl-D>** or the command **exit**. This will log you out of your system. (Logging out is described in more detail in the following section.)
- When you see the prompt

  ```
  Coherent 386 login:
  ```

  type **root**, to log in as the superuser **root**. COHERENT will ask you for the superuser's password; type the password that you assigned to the superuser when you installed COHERENT onto your computer. The Lexicon article on **superuser** describes what the superuser is; as will later sections of this tutorial.
- Once you have logged in as the superuser, type the following command:

  ```
  /etc/shutdown halt 0
  ```

  As its name implies, this command shuts down the COHERENT system. The command will ask you if you really, truly wish to shut down COHERENT; reply 'y', for "yes".
- Now, you can turn the computer off. Or, you can type **<ctrl><alt><del>**, or press the reset button on your computer (should it have one).

  After you have rebooted your computer, just sit back and wait until you receive the **Coherent 386 login:** prompt on your screen.

  If you wish to reboot MS-DOS, watch the computer: wait until you see the computer attempting to read from the floppy-disk drive. At that moment, press the number key that corresponds to the hard-disk sector on which you stored MS-DOS, from 0 to 7. For example, if MS-DOS is kept on partition 2, then press **2** when the computer is attempting to read the floppy-disk drive. Be sure to press the number key that is on the main bank of keys, — *not* the key on the numeric keypad.

That's all there is to it. Shutting down is relatively simple and straightforward; but if you do not take the time to shut COHERENT down properly, you will find that you have destroyed some or all of your data.

By the way, the Lexicon articles on **booting** and **login** describe in detail the processes of booting and logging into your COHERENT system.

Logging Out

As noted above, *logging in* tells COHERENT who you are and that you wish to work with COHERENT for a while. When you have finished working with COHERENT, you must tell COHERENT that you are done for now. This process is called *logging out*.

There are two ways to log out. Each involves typing a special command to the COHERENT prompt. The first way is to type **<ctrl-D>** at the COHERENT prompt. The second is to type the command:

```
exit
```

Each of these commands has the same effect: the COHERENT system flushes all buffers that you “own” and prints the prompt

```
Coherent 386 login:
```

on your screen. At this point, you cannot issue any commands to COHERENT; but you (or someone else) can log into COHERENT from your terminal.

Please note that logging out is *not* the same as shutting down COHERENT. When you shut down COHERENT, you are shutting down the entire system. When you log out, however, you are simply ceasing to work with COHERENT. After you log out, however, COHERENT continues to work on its own: organizing files, exchanging information with other computers via modem, executing programs for users who have logged in via modem or other terminals, and in general making itself useful. If you shut off the computer after you log out, you will damage the file system, just the same as if you shut it off while you were logged in.

The following sections in this tutorial will go into COHERENT’s commands in more detail. All, however, build on the elementary actions presented here: logging into COHERENT; issuing commands; receiving responses from COHERENT; and logging out.

Working With Files and Directories

The *file* and the *directory* are the cornerstones of the COHERENT system. Practically everything you do on the system will involve files: changing files, invoking files, transmitting or receiving files, filling files up or emptying files out. And, directories let you organize masses of files into a rational hierarchy.

This section discusses manipulating files and directories under the COHERENT system. It covers the following:

- What *file* and *directory* mean to COHERENT
- Introduces the commands for manipulating files, directories and their contents
- Discusses more advanced topics, such as creating and mounting new file systems
- Tours the COHERENT file system

This section of the tutorial covers much ground in a relatively brief space. Readers who are new to personal computers should concentrate on the earlier sub-sections, which cover elementary topics; whereas more experienced readers may wish to concentrate on the later sub-sections, which cover the more technical material.

File Names

A *file* is a mass of electronic impulses that is given a name and stored on a disk. Files are given names to make them easy for you to retrieve. COHERENT has rules about how files can be named, to ensure that each file’s name is unique.

The following are examples of legal file names:

```
.profile
File01
cmd.sh
file01
test.c
```

File names are generally made up of upper-case and lower-case letters and numbers. COHERENT, unlike MS-DOS, distinguishes capital letters from lower-case letters; therefore, to COHERENT the file names **File01** and **file01** are different.

Any character can be used to name a file, including a control character. We recommend, however, that you name files using only upper- or lower-case alphabetic characters, numerals, and the punctuation marks '.' or '_'.

The file name must not be more than 14 characters long. If you specify a longer name, characters beyond the 14th will be lopped off and thrown away. For example, COHERENT regards the file names

```
this_is_very_long_file_name_1
```

and

```
this_is_very_long_file_name_2
```

as being identical.

Introduction to Directories

A *directory* is a group of files that have been given a name. Directories let you organize files systematically. This may not seem important now, but as you work with COHERENT you will find that you accumulate hundreds, or even thousands, of files; without system of directories to organize files, you would quickly lose track of what each file held, and find it nearly impossible to find any given file within your system.

Because files are stored within directories, the complete name of a file actually consists of its name plus the name of the directory in which it is stored. This lets COHERENT distinguish files that have the same name but are stored in different directories. COHERENT uses the slash character '/' to distinguish a directory name from a file name; for example, to view the contents of file **junk** in directory **text_files**, you would use the command:

```
cat text_files/junk
```

This system of naming will be described in full in the next sub-section; for the moment, just bear in mind that for COHERENT to find a file, you must tell COHERENT not only the name of the file, but the name of the directory in which it is kept.

When you work with COHERENT, you are always "in" a directory. The directory you happen to be "in" at any given moment is called the *current directory*. The current directory is the one whose files you are working with at this moment. When you type the name of a file and do not mention what directory it is stored in, COHERENT assumes that the file is kept in the current directory. COHERENT includes commands that let you shift from one directory to another.

When you log into COHERENT, COHERENT places you "in" a directory that you "own". This directory is called your *home directory*. You control all of the files in your home directory; it is your "base of operations" for working within COHERENT.

Path Names

As you may have deduced by now, a directory can contain both files and other directories. The directories within a directory may themselves contain both files and directories; which then may contain other files and directories; and so on.

This design of directories branching into other directories, which in turn branch into still other directories, is called *tree structured*. As the tree-metaphor implies, the COHERENT system of directories has a *root directory,* that is, a directory that is not contained in any other directory but from which all other directories descend, directly or indirectly. The name of the root directory is simply:

```
/
```

One subdirectory of the root directory is called **usr**. This subdirectory contains the home directories of all users. Other common paths for home directories are **/u** and **/usr/acct**. To list the names of all user directories, type the command:

```
lc /usr
```

If your login name is **henry**, then the command

```
lc /usr/henry
```

lists the names of the files in your home directory. Please note that in the argument **/usr/henry**, the first slash names the root directory; all subsequent slashes serve simply to separate one directory name from the next.

The name **/usr/henry** is called a *path name*. The term "path name" means the full name of a given file or directory — including all the directories that lead from the root directory to it.

Path names may be full or partial. All full path names begin with **/** for root, and continue with further subdirectory names. Path names that do not begin with a slash are partial; COHERENT automatically prefixes them with the path name of the current directory to make them complete before it uses them.

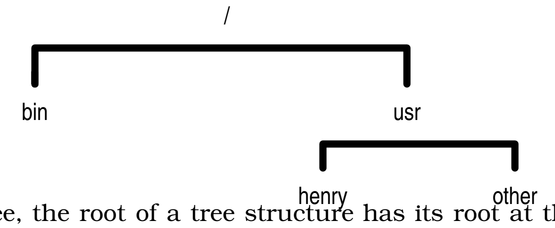
The elements of path names are separated by slashes, so if there were a file in **newdirectory** named **newfile**, you would refer to it as

```
newdirectory/newfile
```

The absence of a beginning slash indicates that the path name begins in the current directory. Thus, if your home directory name is **henry**, then another way to name the path to **newfile** is to type:

```
/usr/henry/newdirectory/newfile
```

The following diagram gives a rough description of the structure of the COHERENT file system:



Please note that unlike a real tree, the root of a tree structure has its root at the top rather than at the bottom. Here, the root directory '/' is at the top of the structure. It contains the directories **bin** and **usr** (among many others). Directory **usr** contains directories **henry** and **other** (again, among many others. These directories can contain many other directories and subdirectories.

In summary, a path name lists all the subdirectories leading from the root directory to the file in question. In the above example, **newfile** is a file in subdirectory **newdirectory**, which in turn is a file in the home directory **henry**, which is further a file in the directory **usr**. The directory **usr** is a file in the master or **root** directory for the system.

You don't need to specify all of this, fortunately, whenever you want to specify a file in a subdirectory. COHERENT assumes that partially specified path names are within the current directory. Therefore, you can specify a subdirectory by specifying the name of the directory first, followed by the rest of the path name.

COHERENT also allows two special abbreviations for directories. The abbreviation '..' always represents the current directory's *parent* directory. In the case of the directory **/usr/henry**, directory **usr** is the parent of directory **henry**. In other words, '..' stands for the directory in which the current directory resides. Every directory in the system except the root directory has a parent. For the root directory, '..' refers to itself.

Another directory abbreviation is '.', which means the *current* directory.

The following sub-sections describe the commands that COHERENT includes for manipulating files and directories. As you work with COHERENT, you will use these commands continually, so it would be worth your while to spend a little time learning them.

ls, lc: Listing Your Directory

This sub-section introduces two of the more commonly used commands: **ls** and **lc**. Both **ls** and **lc** list the files in a directory.

To see how these commands work, presume that your directory has the files created in previous sections and that you did not remove directory **newdirectory**. To list the files in your directory, simply use the command with no parameters:

```
ls
```

This produces a list of files, such as:

```
another
backup
doc1
doc2
file01
file02
newdirectory
stuff
```

The command **lc** also lists file names, but it prints the files and directories separately, in columns across the screen. For example, typing

```
lc
```

gives something of the form:

```
Directories:
   backup newdirectory
Files:
   another doc1 doc2 file01 file02
   stuff
```

If you want to list files in a directory other than your own, name that directory as an argument to the command. For example, **/bin** is a directory in the COHERENT system that contains commands. Type

```
lc /bin
```

and **lc** will print the contents of **/bin**.

Both **ls** and **lc** can take options. An option is indicated by a hyphen '-'. The option must appear before any other argument. For example, to list only the files in the directory for user **carol**, leaving out any directories, use the **f** option with **lc**:

```
lc -f /usr/carol
```

Or, if you type the command

```
lc -f
```

the COHERENT system prints all of the files in the current directory. The following gives the commonly used options to the command **lc**:

-d List directories only, omitting files
-f List files only, omitting directories
-1 List files in single-column format

ls produces a list of file names, one per line, and optionally much more information. To produce all the information, use the **-l** option (note that this is an "el", not a numeral 1):

```
ls -l
```

The following gives a sample of the long list that this option produces. Headings have been added to show the meaning of each column:

Mode	*#*	*Owner*	*Size, Bytes*	*Modification Date*	*Time*	*Name*
-rw-r--r--	1	you	17	Wed Aug 19	17:51	file01
drwxrwxrwx	2	you	32	Wed Aug 19	17:53	backup
-rw-r--r--	1	you	17	Wed Aug 19	17:53	doc1

The meaning of each column will be explained later. For now, note that the last column gives the name of each file, and the fourth column from the left gives the size of each file, in bytes.

cat: Print Contents of a File

The command **cat** opens and prints the contents of a text file — that is, a file of source code, a document, or a message file. For example, to list the contents of file **file01**, type:

```
cat file01
```

This command types the file's contents on the terminal (sometimes also called the *standard output*).

Another use for **cat** — the use from which it gets its name — is to *concatenate* several files on the standard output. For example, the command

```
cat one two three
```

prints the files **one**, **two**, and **three**, one after the other, on your screen.

You can use **cat** to concatenate several files into one file by *redirecting* the standard output into a file. The special character '>' tells COHERENT to redirect the standard output into a file. For example, the command

```
cat one two three >four
```

concatenates files **one two three** into file **four**. **four** need not exist prior to this command; if it does, its contents are replaced by the data redirected into it.

Redirection is a very useful feature of COHERENT that will be used through the rest of this tutorial. The '>' operator also gives an example of the set of operators that can be used with COHERENT commands. These operators, which increase the power of each COHERENT command, will be described in detail later in this tutorial.

more: List Files on the Screen

If the file you list with **cat** is more than 24 lines long, the beginning lines of the file scroll off the screen too quickly for you to read them. To ensure that you see all of the lines in the file, use the command **more**.

more prints a file in 24-line chunks. After it has listed a chunk of text, it pauses and waits for you to press **<space>**. If you call **more** with an option of **-s**,

```
more -s file
```

it will skip all blank lines that are in the text file.

mkdir: Create a Directory

The command **mkdir** creates a new directory. For example, to create a new directory named **newdirectory**, type the following command:

```
mkdir newdirectory
```

If you follow this command with **lc**, it lists your regular files, but it also lists **newdirectory** separately as a directory:

```
Directories:
    newdirectory
Files:
    file01      file02
```

To refer to any files in **newdirectory**, use its name in specifying the path name.

Now, create a file in the new directory:

```
cat >newdirectory/newfile
lines to be
contained in newfile
<ctrl-D>
```

This command copies lines to the file described by the partial path name **newdirectory/newfile**.

cd: Change Directory

The command **cd** changes the current working directory. For example, the command

```
cd newdirectory
```

moves you into directory **newdirectory** that you created in the previous sub-section. Now, if you type the command **lc**, to show the contents of the current directory, it will show the following:

```
Files:
    newfile
```

To return to the previous directory, use the command:

```
cd ..
```

As noted earlier, the abbreviation '..' always indicates the current directory's parent directory.

pwd: Print Working Directory

The command **pwd** prints the name of the current, or *working*, directory. For example, if your login name is **henry**, then if you type

```
pwd
```

you will see:

```
/usr/henry
```

Now, use the **cd** command to switch to directory **newdirectory**, as follows:

```
cd newdirectory
```

When you type

```
pwd
```

you will see:

```
/usr/henry/newdirectory
```

Finally, use the **cd** command to return to the previous directory, as follows:

```
cd ..
```

When you type

```
pwd
```

you now see:

```
/usr/henry
```

If you are ever unsure what directory you are in, use the **pwd** command.

mv, cp: Move and Copy Files

The command **mv** moves files. You can move a file from one name to another within the current directory (in effect rename the file), or you can move a file from one directory to another. **mv** takes two parameters: the first names the file to be moved; the second names either the new name that you are giving to the file, or the directory into which you are moving the file.

For example, to move file **file01** into directory **newdirectory**, type:

```
mv file01 newdirectory
```

To see where **file01** is now, type the following command:

```
lc newdirectory
```

The result is:

```
Files:
    newfile
```

To move **newfile** back into the current directory, use the command:

```
mv newdirectory/newfile .
```

Remember, the abbreviation '.' always stands for the current directory.

As noted above, the **mv** command can also be used to rename files within the current directory. For example, to change the name of **newfile** to **oldfile**, use the following command:

```
mv newfile oldfile
```

If the current directory already has a file named **oldfile**, it will be thrown away and replaced with the file that used to be named **newfile**.

The command **cp** copies a file. This command has two parameters: the first names the file to be copied, and the second names the file or directory into which it is to be copied. For example, to copy **oldfile** in the current directory back into **newfile**, use the following command:

```
cp oldfile newfile
```

If **newfile** already exists, it will be replaced by a copy of **oldfile**.

If you wished to copy **newfile** into directory **newdirectory**, use the command:

```
cp newfile newdirectory
```

Now, when you type the command

```
lc newdirectory
```

you will see:

```
Files:
    newfile
```

As you can see, **newfile** has been copied into **newdirectory**. If **newdirectory** had already contained a file called **newfile**, that file would have been replaced with the newer **newfile** being copied into **newdirectory**.

The following example summarizes what's been presented so far about files and directories. For purposes of the example, assume that your login name is **henry**, and that you have in your home directory files **doc1** and **doc2** that you wish to back up for safekeeping.

Before you can back up these files, you must first create them. First, use the command **cat** to create file **file01**, as follows:

```
cat >doc1
a few
lines of
text
<ctrl-D>
```

Likewise, create file **doc2**:

```
cat >doc2
second file
with some text
<ctrl-D>
```

(Don't forget that **<ctrl-D>** means to hold the control key down and simultaneously type **D**.)

The command **lc** will now show you the files and directories in your current directory:

```
Directories:
    newdirectory
Files:
    doc1    doc2    newfile    oldfile
```

The next step is to create the directory to hold the back-up copies. To help remind yourself what the directory is for, name it **backup**.

```
mkdir backup
```

Now, **lc** shows you:

```
Directories:
    backup      newdirectory
Files:
    doc1    doc2    newfile    oldfile
```

The next step is to use **cp** to copy your files into **backup**:

```
cp doc1 backup
cp doc2 backup
```

After you issue these commands, **lc** still says:

```
Directories:
    backup      newdirectory
Files:
    doc1    doc2    newfile    oldfile
```

However, if you list the contents of subdirectory **backup**

```
lc backup
```

you will see:

```
Files:
    doc1 doc2
```

The files have been successfully copied into the back-up directory.

For a full description of these commands and the options available with each, see their respective entries in the Lexicon.

rm, rmdir: Remove Files and Directories

The command **rm** removes a file. For example, if you wish to remove file **doc2** in directory **backup**, type the following command:

```
rm backup/doc2
```

After typing this command, use the command **lc** to show the contents of directory **backup**, as follows:

```
lc backup
```

You should see:

```
Files:
    doc1
```

As you can see, file **doc2** has been removed.

You can remove several files at once, simply by listing them on the **rm** command's command line. For example:

```
rm file01 file02
```

removes files **file01** and **file02**.

Note that once you remove a file with **rm**, it is gone forever. The COHERENT system does not warn you if you **rm** several files at once; it will assume that you know what you're doing and carry out your command silently. For this reason, be careful when you use the **rm** command, or you may receive a rude surprise.

You cannot use the command **rm** to remove a directory. COHERENT does this to help prevent you from wiping out an entire file system with one simple **rm** command. To remove a directory, use the command **rmdir**. For example, to remove the directory **newdirectory**, type:

```
rmdir newdirectory
```

Note that before you can delete a directory, that directory must not have any files or directories in it. If you try to remove a directory that has files or directories in it, COHERENT will print an error message on your screen and refuse to remove the directory.

For a full description of these commands and the options available with each, see their respective entries in the Lexicon.

du, df: How Much Space?

Files occupy space on your hard disk. (A corollary to Parkinson's law states that files expand to fill the disk allotted to them.) It is somewhat disconcerting to attempt to save a large file, only to find that you have run out of disk space. To help you manage your hard disk, COHERENT includes the commands **du** and **df**.

The disk-usage command **du** tells you how much disk space the files in the current directory occupy. If the directory has sub-directories, these are listed separately. **du** prints disk usage in blocks; each block is 512 bytes (half a kilobyte).

The disk-free command **df** tells you how many blocks are left free on your disk. By default it prints information only about the file system you are now in.

If you find that you are running low on disk space, you must free up some space. You can do that by removing files you no longer need; by *compressing* files that you do not use often; or by backing files up to floppy disk and then removing them. We have already described how to remove files. Look in the Lexicon entry for the command **compress** for information on how to compress and uncompress files. Following sections in this tutorial will describe how to copy files to floppy disk.

For more information on these commands, see their respective entries in the Lexicon.

ln: Link Files

COHERENT allows a file to have more than one name. When you create a file, you give it a name; COHERENT *links* the name you give the file with its internal system of managing files. (For more information on how COHERENT identifies files, see the Lexicon entry for **i-node**.) COHERENT allows you to give a file more than one name; another way of expressing this is to say that you can give a file *multiple links*.

To create a new link to an existing file, use the command **ln**. This command takes two arguments: the first names the file to which you wish to give a new link, and the second gives the name that you wish to link to that file. If the name you are linking to a file is already being used by a file, COHERENT will not let you link the file to that name.

For example to link the file **doc1** to the name **another**, use the following command:

```
ln doc1 another
```

The "new" file has the same data in it as the "old" file; in fact, the names **doc1** and **another** are synonyms for the same file.

The next point is somewhat subtle. When you use the command **rm** to remove a file, what you are actually doing is breaking the link between that file and its name. The file is not actually removed from disk until all links are broken between it and all of its names. In the above example, if you use the command

```
rm another
```

to remove the file **another**, the file **doc1** remains in existence, and the data to which the names **another** and **doc1** pointed remains on the disk. If you then use the command

```
rm doc1
```

to remove **doc1**, then you will have broken all links between that file and the COHERENT system, and COHERENT removes it from the disk.

Links are useful if you wish a file to be used in two different contexts but have the same data. For example, if you use file **doc1** in two different manuscripts, you can create links to the file in two different directories, one for each manuscript. Thus, any changes you make to the file under either its names appear automatically in both manuscripts.

As always, see the Lexicon for a full description of the **ln** command.

File Permissions

As you recall, the command **ls -l** prints a mass of information about each file. The following repeats the information that appeared when you typed **ls -l**:

Mode	*#*	*Owner*	*Size, Bytes*	*Modification Date*	*Time*	*Name*
-rw-r--r--	1	you	17	Wed Aug 19	17:51	file01
drwxrwxrwx	2	you	32	Wed Aug 19	17:53	backup
-rw-r--r--	1	you	17	Wed Aug 19	17:53	doc1

Column 3 names the owner; in this example, **you** represents your login name, whatever you have set it to. Column 4 gives the size of the file, in bytes. Columns 5 through 7 give the day of the week and the date on which the file was last modified. Column 8 gives the time the file was last modified or, if the file was last modified more than a year ago, the year it was last modified. Column 9 gives the name of the file.

Column 1 gives the *mode* of the file. The mode summarizes the *permissions* attached to this file.

Before going further, the concept of file permissions should be reviewed. COHERENT is a multi-user operating system, which means that more than one person can log into the system, walk through its file system, execute commands, and manipulate files. Every user has files that she "owns" — that is, that she has created and that she wishes to protect against being altered or removed by others. After all, it would be disconcerting if you were to log into your system, only to find that some of your key files had been trashed by another user, without your knowledge or permission.

The COHERENT system protects files by its system of file permissions. Permissions have two aspects: the *type* of permission, and the *scope* of permission. There are three *types* of permission:

read permission
Permission to read a file.

write permission
Permission to write into a file.

execute permission
Permission to execute a file, assuming that file contains executable code instead of text.

Likewise, there are also three types of *scope:*

user The permissions extended to the owner of the file.

group The permissions extended to the group of users to which the owner belongs. For more information on what *group* is, see the Lexicon entry for **group**.

other The permissions extended to all other users.

The *mode* column describes all permissions attached to a file. It also gives other information about a file, such as whether the file is a directory. Taking the entry for file **file01** as an example, we see:

```
1 2  3  4   # Owner            Size      Date         Time    File name
-rw-r--r--  1 you         17 Sat Aug 15 17:20  file01
```

As you can see, the mode field is divided into four subfields, in this example labelled '1' through '4'.

Subfield 1 indicates whether this file is a directory. If the file were a directory, this would contain a **d**; otherwise, it contains a hyphen.

Subfields 2 through 4 describe the type of permission extended to, respectively, the owner, the owner's group, and other users. Each subfield consists of three characters. The first character indicates whether the file is readable; if it is, then the character is an 'r'; otherwise, it's a hyphen. The second character indicates whether the file is writable; if it is, then the character is a 'w'; otherwise, it's a hyphen. The third character indicates whether the file is executable; if it is, then the character is an 'x'; otherwise, it's a hyphen.

In the above example, file **file01** has permissions:

```
-rw-r--r--
```

These grant read and write permission to its owner, read permission to the other members of the owner's group, and read permission to all other users.

The COHERENT system has a set of default permissions that it applies to every file when it's created. To change this default set of permissions, use the command **umask**. For information about this command, see its entry in the Lexicon. To change the permissions of an existing file, use the command **chmod**, as described in the following sub-section.

chmod: Change File Permissions

To change the mode of a file, use the change-mode command **chmod**. For example, to protect file **doc1** in directory **backup** from being overwritten, use the command:

```
chmod -w backup/doc1
```

where the **-w** means "remove write permission" and is followed by the file name. Henceforth, if you try to write into this file, the COHERENT system will refuse to do so and will print an error message on your screen.

To allow other users to read the backup file **doc2**, type:

```
chmod o+r backup/doc2
```

where the letter **o** signifies "other users", and the **+r** tells **chmod** to grant read permission.

To see the new set of permissions, type the command:

```
ls -l backup
```

As you can see, the mode string has changed from what it was above.

Directory access permissions are similar to file access permissions in that they can easily be changed via command **chmod**. However, the permission bits have different meanings for directories. Permitting reads on a directory allows the user to see the contents of the directory via commands such as **lc** or **ls**; permitting execution on a directory allows access to the files in the directory; and permitting writes on a directory allows the user to create or delete files in the directory, regardless of the permissions on the actual file. The latter causes the most difficulty for new users since they mistakenly associate file deletion permissions with the actual file rather than with the directory that contains the file.

Creating and Mounting a File System

Earlier, we described how the COHERENT system consists of a tree of directories; and how that tree branches from the root directory '/'. This is a useful description, and true as far as it goes; but the full situation is a little more complex.

The tree of COHERENT directories in fact consists of any number of *file systems,* each of which exists on its own physical device. A *physical device* may be a partition on your hard disk, a floppy disk, or even a chunk of RAM.

The COHERENT system contains a suite of commands that let you create a new file system on a physical device, and graft (or *mount*) that new file system onto the COHERENT directory tree. The following few sub-sections will walk you through the steps of creating a new file system on a floppy disk and mounting it onto your existing COHERENT directory tree. These descriptions may be a bit too advanced for beginners; but most users will find them to be interesting and helpful.

fdformat: Format a Floppy Disk

The first step in creating our new file system is to format a floppy disk. The command **fdformat** formats a diskette. When a diskette is formatted, COHERENT writes information on each track that makes it possible for the floppy disk to hold a file system.

fdformat uses the following syntax:

`/etc/fdformat` *device*

where *device* is the name of the device to be formatted. To format a high-density, 5.25-inch diskette, use the command:

```
/etc/fdformat /dev/rfha0
```

To format a high-density, 3.5-inch diskette, type:

```
/etc/fdformat /dev/rfva0
```

To format a low-density, 5.25-inch diskette, type:

```
/etc/fdformat /dev/rf9a0
```

For this example, we'll assume that you have a high-density, 5.25-inch floppy disk. Insert into drive 0 (that is, drive A) of your computer, and type the command:

```
/etc/fdformat -v /dev/rfha0
```

The **-v** option to **fdformat** tells it to verify that the disk is sound. This option means that the command will take longer to execute, but in the long run it's worth it as it will ensure that you do not waste time to trying to copy data onto a flawed disk. For details on the command **fdformat**, see its entry in the Lexicon.

When this command has finished executing, leave the floppy disk in drive 0.

mkfs: Create a File System

The command **mkfs** creates a file system on a physical device. This command has the following syntax:

`/etc/mkfs` *special proto*

special names the physical device on which the file system is to be built. *proto* is either a number or a file name. If it is a number, **mkfs** builds a file system of that size in blocks.

For our example, type the command:

```
/etc/mkfs /dev/fha0 2400
```

This command writes a file system onto device **/dev/fha0**, which in this case represents the floppy disk in drive 0 that we just formatted. The number 2400 represents the number of blocks that fits onto such a disk. Please note that the above example is for a 5.25-inch, high-density floppy disk. For directions on how to create a file system on a floppy disk of different size or density, see the Lexicon articles for **floppy disks** or **mkfs**.

If *proto* is not a number, **mkfs** assumes that it is a prototype file. The command **badscan** scans a physical device for bad blocks and writes such a prototype file for you. Prototype files are beyond the scope of this example; but for information on them see the Lexicon entry for **badscan** or the Lexicon entry for **floppy disks**. The latter article summarizes all the ways in which floppy disks are used by the COHERENT system.

mount: Mount a File System

Now that you have formatted your floppy disk and built a file system on it, you can *mount* the newly created file system. *Mounting* grafts this device's file system onto the COHERENT system's directory tree. Thereafter, you can write files onto that device, read them, remove them, or do anything else that you wish with that device and its contents.

mount has the following syntax:

`/etc/mount` *device directory*

device names the physical device whose file system is to be mounted. *directory* names the *base directory* for that file system. The base directory is the directory by which the file system is accessed. For example, directory **/usr** is the base directory for the file system that holds all users' home directories. We'll describe base directories a little further in a few paragraphs.

For purposes of our example, type the following command:

```
/etc/mount /dev/fha0 /f0
```

This mounts the file system on the disk in drive 0 onto base directory **/f0**.

The base directory by convention is a directory in the root directory '/'. You do not have to do this, however. For example, if your user name was **henry** and you wished to mount the file system on the floppy disk in your home directory, you could type:

```
/etc/mount /dev/fha0 /usr/henry/backup
```

This will mount the file system on the floppy disk onto directory **/etc/henry** and name its base directory as **backup**. Note that if directory **backup** already existed in directory **/usr/henry**, its contents will be inaccessible until you unmount the file system on the floppy disk. Unmounting is discussed in the following sub-section.

For more information on mounting a file system, see the Lexicon article **mount**.

Using a Newly Mounted File System

Now that you have created and mounted a file system, you can use it like any other directory. To see how this works, type the following command:

```
cat >/f0/testfile
Here's some text we're writing onto the
newly mounted file system on a floppy disk.
<ctrl-D>
```

Here you can use the **cat** command to write some text into file **testfile**, which lives on the floppy disk you just mounted. To see that this text has been written there, type:

```
cat /f0/textfile
```

You should see the floppy-disk drive whirl briefly, and the following appear on your screen:

```
Here's some text we're writing onto the
newly mounted file system on a floppy disk.
```

You can now use this file system like any other, even though it lives on a floppy disk rather than your hard disk. As you can see, this is an easy way to extend the size of your COHERENT system's file system.

umount: Unmount a File System

Finally, when you have finished working with a file system, you must use the command **umount** to un-mount it. This command prunes the file system on a given physical device from the COHERENT system's directory tree. You will use this command frequently as you use floppy disks.

umount takes one argument: the name of the physical device being unmounted. In our example, the command

```
/etc/umount /dev/fha0
```

unmounts the file system on the high-density, 5.25-inch floppy disk insert into drive 0 (that is, drive A) on your computer.

Under unsophisticated operating systems like MS-DOS, you can insert or remove floppy disks without giving the matter a second though. The COHERENT system, however, uses a complex set of buffers to speed the reading and writing of information to the floppy disk; for this reason, if you simply yank a floppy disk out of its drive, all of the information in the COHERENT system's buffers will be lost. Worse, if you yank out a floppy disk and insert a COHERENT-formatted floppy disk, the COHERENT system will write the data in its buffers onto that new floppy disk — and probably destroy its file system in the process. Unmounting a file system tells the COHERENT system to flush all information in its buffers and write it onto the disk.

To emphasize this point, please read the following carefully:

If you mount a floppy disk, you must use the **umount** *command to unmount it before you remove the disk from its drive. If you do not, data will be destroyed.*

This concludes the discussion of how to mount create a file system, mount it, and use it. See the Lexicon article **floppy disks** for further information on how to do this task.

The following two sub-sections discuss how to check a file system, to ensure its integrity.

fsck: Check a File System

The command **fsck** checks a file system, to ensure its integrity. For example:

```
fsck /dev/root
```

where **/dev/root** is a disk device, checks the file system located on device **/dev/root**.

If possible, you should **umount** the file system before you check it. You cannot **umount** the root file system. If you can't unmount it, be sure that no other users are on the system (i.e., that you are in single-user mode), then reboot the system immediately *without* performing a **sync**. If other users are creating or expanding files while the file systems are being checked, **fsck** will report false errors.

If **fsck** finds any discrepancies, it writes appropriate messages onto the console (that is, the screen directly plugged into your computer). An absence of messages indicates that there are no problems with the file system. The appendix to this manual gives all of **fsck**'s error messages, and suggests how you should respond to each.

COHERENT's boot routines run **fsck** automatically, and will rerun it if necessary to fix problems with the file system. For more information on **fsck**, see its entry in the Lexicon.

Devices, Files, and Drivers

The next few sub-sections introduce the topic of special files and devices. You brushed this topic in the earlier section that described how to format and mount a file system on a floppy disk; the following few sections go into it more systematically. Beginners will probably find that much of this sub-section is mystifying, but experienced users and ambitious beginners probably will find much of value here.

To begin, the COHERENT system is designed to provide device-independent I/O. Devices and files are handled in a consistent way. Each I/O device is represented as a *special file* in directory **/dev**. For example, if your system has a line printer device named **lp**, you can list a file, named **prog** for example, on the printer by saying:

```
cat prog >/dev/lp
```

Another example is to copy the file **prog** with the **cp** command to your terminal:

```
cp prog /dev/tty
```

There are two types of special files represented in **/dev** and when you list **/dev** with **lc** it will separate them.

The first type is a *block special* file. This type includes disks and magnetic tape. These devices are read and written in blocks of 512 bytes, and can be randomly accessed. (As a practical note, note that magnetic tape can be read in a random fashion only by positioning backwards and forwards one record at a time; disks can be read or written in a totally random fashion.)

The I/O to and from block devices is buffered to improve overall system performance. When a program writes a block of data, the data are held in a buffer to be written at a later time. If the same block is read twice in a row, the data for it is still available in memory and do not have to be fetched from the physical device.

A special program named **/etc/update** forces all buffered data to the physical device periodically by calling the command **sync** to protect against losing data in the case of an accident, such as a power failure. If you must bring the system down, you must force the latest data to be written by typing the command **sync**.

Character-Special Files

The second kind of special file is called a *character-special* file. Included in this class are devices that are not block special: terminals, printers, and so on. Disks and tapes can also be treated as character special files. For every block special file for a disk, such as

```
/dev/at0c
```

there is usually a character-special file:

```
/dev/rat0c
```

Character-special files are sometimes called *raw* files, hence the prefix **r** in **rat0c**. A raw file has no buffering or other intermediate processing performed on its information. This difference is an efficient benefit to commands such as **dump** and **fsck**, which do their own buffering.

tty Processing

One special set of devices has other processing — the **tty** or terminal files. A terminal-special file with this special processing is called a *cooked* device. The processing includes handling the **kill**, **erase**, **interrupt**, **quit**, **stop**, **start**, and **end-of-file** characters. Processing can be disabled with the command **stty** so the program deals with the raw device. However, using a raw **tty** device generally has negative effects on performance of the COHERENT system.

A Tour Through the File System

Our introduction to COHERENT's system of files and directories concludes with a tour of the COHERENT file system. Much of this material has been described earlier.

General File System Layout

The base of the file system is the **root** directory, whose name is simply:

```
/
```

Most of the files in the root are directories. To list the files in the **root** directory, type:

```
lc /
```

/bin

Most of the commonly used commands are programs contained in **/bin**, such as the command **lc** used in the above example. Foreign commands, such as MicroEMACS and **kermit**, are placed in directory **/usr/bin**.

The shell does not automatically look in **/bin** for commands, but consults the variable **PATH** to determine where commands are to be found. A typical value for **PATH** is:

```
/bin:/usr/bin:.
```

This tells the shell to look for commands in three places (in this order): **/bin**, **/usr/bin**, and finally **.**, the current directory. The shell does not consult **PATH** if the command contains one or more **/** characters, indicating a complete or partial path specification.

/dev

Devices in the COHERENT system are accessed through files in the directory **/dev**. If there is a line printer available on the system named **lp**, you can print characters from a file named **testdata** by typing the command:

```
cat testdata >/dev/lp
```

All devices on the system are represented in the **/dev** directory. Note that it is not recommended you access devices directly, but use the COHERENT system's utilities that *spool* files to them. This will prevent two users attempting to write material to a device simultaneously, and so garbling the output. For example, to access the line-printer device, use the spooler **lp**. See the Lexicon's entries on **printer** and **device drivers**.

/drv

A unique feature of the COHERENT system is the concept of loadable device drivers. This feature lets COHERENT system programmers write their own device drivers without modifying the rest of the system. Drivers can be unloaded, modified, and reloaded without halting and rebooting the system. Loadable drivers are kept in directory **/drv**. To load a driver, type:

```
/etc/drvld /drv/driver
```

where *driver* is the driver to load. See the Lexicon's entry on **drvld** for more information.

/etc

Several commands that you will use in your role as system administrator are kept in directory **/etc**. These are described in detail elsewhere in this guide. They include commands for system accounting, booting the system, mounting the system, create file systems, and control system time.

Also in **/etc** are several data files used in system administration. These include **/etc/passwd**, the file containing user names, ids, and passwords; news files; and file **/etc/ttys**, which describes the properties of each user terminal attached to the system.

/lib

The COHERENT system provides many useful functions for performing input and output (I/O) and mathematics, for use in your C programs. These and other libraries, along with the phases of the C compiler itself, are kept in directory **/lib**. This directory includes files containing standard system calls, standard I/O, and mathematical routines such as **sin**, **cos**, and **log**.

/usr

The directory **/usr** contains user directories, along with a few system directories.

/usr/adm contains additional information of interest to the system administrator.

/usr/bin contains commands that were not entirely created by Mark Williams Company.

/usr/games contains computer games. **/usr/games/lib/fortunes** holds a set of *bon mots*; the game **fortune** selects one at random and prints it on your screen. A call to this game can be placed in a user's **.profile**, so he will see a new fortune each time that he logs on. To add fortunes of your own, just edit the file **/usr/games/lib/fortunes**.

The directory **/usr/include** contains header files for C programs, such as **stdio.h**. Other header files define formats of files and other important data structures in the system.

/usr/lib contains the macro files **ms** and **man** used the **nroff** text processor; the unit conversion tables for the command **units**; and the file **/usr/lib/crontab** used to hold commands for **cron**. This directory also holds the C libraries.

/usr/man contains manual sections referenced by the commands **man** and **help** commands.

/usr/msgs stores messages displayed by the command **msgs**.

/usr/pub contains public files, such as telephone numbers and a copy of the ASCII table.

/usr/spool contains information for line-printer spooling, and mail that has not yet been delivered.

/u

In some systems, users' directories are placed on a separate device to save space. Because a separate device has a separate file system, the directory on that device is called **/u**.

Files: Conclusion

This concludes this tutorial's discussion of files and directories. The rest of this tutorial introduces COHERENT's suite of commands, and discusses topics of special interest to persons who are administering COHERENT systems.

Introduction to COHERENT Commands

This section introduces COHERENT's commands. The COHERENT system comes with more than 200 commands, which perform a variety of work, from formatting text, to editing files, to performing low-level administration of the system. The commands that manipulate files and directories were introduced in the previous section; there are, however, many other varieties of commands, many of which will be introduced here. To begin, we'll introduce the COHERENT system's master command, the *shell*.

The Shell

When you type commands into the COHERENT system, it appears that you are communicating directly with the computer. This is not exactly true, however. When you type into the COHERENT system, you are actually working with a special COHERENT program, the shell. This program reads, interprets, and executes every command that you type into the system. The shell can also interpret, expand, and otherwise flesh out what you type; this is done to help spare you unnecessary typing, and to permit you to assemble powerful commands with only a few keystrokes.

Please note, in passing, that the COHERENT system comes with two shells: the Korn shell **ksh** and the Bourne shell **sh**. These shells have somewhat different features. The descriptions in this section assume that you are using **sh**, which is COHERENT's default shell.

The shell is so powerful that mastering it is a major accomplishment; however, you can take advantage of much of what the shell offers by learning a few simple commands and procedures.

This section introduces some commands commonly used by COHERENT users. For more information on these or other commands see **help** and **man**. Also, consult the Lexicon.

Please note the following special punctuation characters:

```
* ? [ ] | ; { }
( ) $ = : ` ' " < > << >>
```

These characters have special meaning to the shell, and typing them can cause the shell to behave quite differently from what you may expect. Do not use these characters until you have read the following section, which discusses their use, or until they are presented in examples.

Redirecting Input and Output

Most COHERENT commands write their output to the *standard output* device, which is normally your terminal's screen. For example, **who** prints on your terminal the name of each user currently logged into your COHERENT system:

```
who
```

By using the special character **>**, you can redirect the output of **who** into a file. The command

```
who >whofile
```

writes this information into **whofile**. The operator **>** tells COHERENT to *redirect* the standard output. Later, you can list the information on your terminal using **cat**:

```
cat whofile
```

Once the information is in a file, you can process it in other ways. For example

```
sort whofile
```

sorts the contents of **whofile** and prints the results on your screen. In this way, you can display the users' names on your terminal in alphabetical order.

You can also redirect the *standard input* to accept input from a file rather than from your terminal. To redirect the standard input, use the special character **<** before the name of the file that you want read as the standard input. For example, the command **mail** sends electronic mail to another user; normally, it "mails" what you type on the standard input, but you can use '<' to tell it to mail the contents of a file instead.

```
mail fred < whofile
```

mails the contents of **whofile** to user **fred**.

Pipes

The *pipe* is an important feature of the COHERENT system. Pipes allow you to hook several programs together by redirecting the output of one into the input of the next. A pipe is represented by the character '|' in the command line.

Most COHERENT programs are written to act as *filters*. A filter is a program that reads its input one line at a time or one character at a time, performs some transformation upon what it has read, and then writes the transformed data to the standard output device. You can easily perform complex transformations on data by hooking a number of simple filters together with pipes. Consider, for example, the command:

```
who | sort
```

Here, the command **who** generates a list of persons who are logged into the system. The output of **who** is then piped to the program **sort**, which sorts the list of users into alphabetical order and prints them on the standard error device.

The power and flexibility of the COHERENT operating system owes much to the pipe.

Superuser

A special user in the COHERENT system, called the *superuser*, has privileges greater than those of other users. The superuser can read all files (except encrypted files) and execute all programs. You must be logged in as the superuser during certain phases of your work as system administrator.

There are two ways to access the COHERENT system as the superuser. The first is to login under the user name **root**. When the system prompts

```
Coherent 386 login:
```

reply:

```
root
```

This automatically makes you **superuser**. To remind you that you are superuser, the COHERENT system prompts you with **#** instead of the usual **$**.

The second way to acquire the privileges of superuser is to issue the command

```
su
```

when you are logged in as a user other than **root**. You must have privileges to access **root** to do this, and you must know the password for **root**. When you type

```
<ctrl-D>
```

in this mode, COHERENT returns you to your previous identity.

To be the superuser for only one command, use the form of the command

su root *command*

command is the command to be executed as superuser. For example, to edit the message of the day file **/etc/motd** if you are not the superuser, type

```
su root me /etc/motd
```

When you finish using MicroEMACS, your original user id will be unchanged.

To limit access to privileged resources, the COHERENT system requires users to enter *passwords* before being granted that privilege. Users may be required to enter passwords before logging in.

If the **root** user has a password, you will be prompted for it. If you do not enter it correctly, the system will tell you

```
Sorry
```

and not allow you to become the **superuser**.

It is normal practice to protect access to superuser status by setting the password. If you are the only user of your COHERENT system, or if you deeply trust all other users, you do not have to do so. However, because the superuser can perform any sort of mayhem on your system, it is advisable to set the password, especially if outsiders can dial into your system via modem.

vsh: The Visual Shell

Some users prefer to work visually rather than type explicit commands on a command line. For these users, COHERENT offers the command **vsh**, its *visual shell*.

vsh does *not* give you a true windowing system, like X or Microsoft Windows; nor does it use a mouse. However, **vsh** does give you a visual desktop for your files and commands. You can use the arrow keys on your terminal or console to select a command or a file; execute commands with one keystroke; program your function keys to execute commands automatically; and in general make COHERENT easier to use.

vsh is described in full in its Lexicon entry. However, to get the flavor of **vsh**, try the following exercises:

- Before you begin, make sure that COHERENT knows what kind of terminal you are working at. To check that type the command:

  ```
  echo $TERM
  ```

  If you are working at your computer's console, you should **ansipc** on your screen; whereas if you have logged in from an IBM PC plugged into your computer's serial port, you should see the response **vt100**. If you do not see the correct response, do *not* do the following exercises, because all you will on your screen is a jumble.
- If your terminal's type is set correctly, invoke the visual shell by typing **vsh** on your command line.
- In a moment, **vsh** draws its desktop on your screen. You will see five windows: Two long, narrow windows across the top of your screen; a big window on the left; and two small windows stacked on top of each other on the right. The top narrow window has a number of commands in it; the large window on the left displays all of the files in the current directory. The top file displayed in the large, left window is highlighted.
- The two stacked windows on the right of the screen give information about your system. They give, for example, the name of your system, the number of bytes in the current directory, your login identifier, whether you have mail waiting for you, and other information.
- Press the arrow key ↓. The bar of highlighting moves to the next name displayed in the large, left window. The scroll bar in the large, left window also creeps down a little. Now, press the arrow key ↑. The highlighting bar moves up again.
- Press **E** (for Exit). **vsh** opens a pop-up window and asks if you want to really exit. Press `(¢)` to accept the underlined option, 'y'. When you do so, **vsh** then exits and returns you to the COHERENT command line.

vsh is a powerful command that makes COHERENT much easier for the average user. See its Lexicon entry for details on its features and how to use it.

Manipulating Text Under COHERENT

The COHERENT system includes a number of commands and utilities with which you can process text. The phrase *process text* means to edit it and prepare it for printing.

MicroEMACS: Text Screen Editor

COHERENT includes a full-featured screen editor, called MicroEMACS. MicroEMACS allows you to divide the screen into sections, called *windows*, and display and edit a different file in each one. It has a full search-and-replace function, allows you to define keyboard macros, and has a large set of commands for killing and moving text.

Also, MicroEMACS has a full help function for C programming. Should you need information about any macro or library function that is included with COHERENT, all you need to do is move the text cursor over that word and press a special combination of keys; MicroEMACS will then open a window and display information about that macro or function.

For a list of the MicroEMACS commands, see the Lexicon entry for **me**, the MicroEMACS command. A following section of this manual gives a full tutorial on MicroEMACS. In the meantime, however, you can begin to use MicroEMACS by learning a half-dozen or so commands.

To invoke MicroEMACS, type the command

```
me hello.c
```

at the COHERENT prompt. This invokes MicroEMACS to edit a file called **hello.c**. Now, type the following text, as it is shown here. If you make a mistake, simply backspace over it and type it correctly; the backspace key will wrap around lines:

```
main()
{
        printf("hello, world\n");
}
```

When you have finished, *save* the file by typing **<ctrl-X><ctrl-S>** (that is, hold down the control key and type 'X', then hold down the control key and type 'S'). MicroEMACS will tell you how many lines of text it just saved. Exit from the editor by typing **<ctrl-X><ctrl-C>**.

Now, re-invoke MicroEMACS by typing

```
me hello.c
```

The text of the file you just typed is now displayed on the screen. Try changing the word **hello** to **Hello**, as follows: First, type **<ctrl-N>** That moves you to the *next* line. (The command **<ctrl-P>** would move you to the *previous* line, if there were one.) Now, type the command **<ctrl-F>**. As you can see, the cursor moved *forward* one space. Continue to type **<ctrl-F>** until the cursor is located over the letter 'h' in **hello**. If you overshoot the character, move the cursor *backwards* by typing **<ctrl-B>**.

When the cursor is correctly positioned, delete the 'h' by typing the *delete* command **<ctrl-D>**; then type a capital 'H' to take its place.

With these few commands, you can load files into memory, edit them, create new files, save them to disk, and exit. This just gives you a sample of what MicroEMACS can do, but it is enough so that you can begin to do real work.

Now, again *save* the file by typing **<ctrl-X><ctrl-S>**, and exit from MicroEMACS by typing **<ctrl-X><ctrl-C>**.

Just as a reminder, the following table gives the MicroEMACS commands presented above:

<ctrl-N>	Move cursor to the *next* line
<ctrl-P>	Move cursor to the *previous* line
<ctrl-F>	Move cursor *forward* one character
<ctrl-B>	Move cursor *backward* one character
<ctrl-D>	*Delete* a character
<ctrl-X><ctrl-S>	*Save* the edited file
<ctrl-X><ctrl-C>	Exit from MicroEMACS
<ctrl-Z>	Save a file and exit

Note that on some terminals, the arrow keys will not work. Note, too, that some remote terminals may have trouble using **<ctrl-S>**, if they use XON/XOFF to control flow. In this case, use **<ctrl-Z>** instead.

For more information, see the tutorial for MicroEMACS included with in this manual.

pr, prps, lp: Print Files

The command **lp** prints files for you, making sure that your request does not conflict with other uses of the printer. To print a file, type the command

lp *file*

substituting the name of the file to be printed for "file". If you don't name a file on the command line, **lp** prints what it receives from the standard input. Thus, you can use **lp** in pipes; this allows you to print immediately matter that you type on your keyboard.

lp will take your file and try to print it on any printer you have plugged into your computer's parallel port. If you do not have a printer plugged in, or if it is not turned on, **lp** will hold onto your files until the printer becomes ready; it will wait days, if necessary, until the printer becomes available.

lp is also intelligent enough to handle requests from several different users: if more than one user wants to print a file, **lp** will print them one at a time. In this way, the COHERENT system lets several users share one printer.

lp does nothing to the file other than print it. This means that it does not attach page heading to a file, nor does it break up the file into page-sized chunks. Another command, **pr**, does this for you. It paginates the standard input, giving a header with date, file name, page number, and line numbers. The paginated output appears on the standard output.

To print a paginated file on the line printer, type:

pr *file* | lp

Note the use of the pipe '|', which passes the output of **pr** as input to **lp**.

The command **prps** is like **pr**; however, it writes a file in the PostScript page-description language, suitable for printing on a PostScript printer. If you have PostScript on your system, you should use **prps** instead of **pr** to paginate text for printing.

COHERENT has many more features for printing files than can be covered here. For more details on **lp** and on how to print text, see the Lexicon entry for **printer**.

nroff, troff: Text Formatters

The commands **nroff** and **troff** format text for display or printing. They are, in fact, text-formatting languages: you type commands into your text file, and **nroff** or **troff** interprets the commands to format the text in the manner that you want.

nroff and **troff** differ in the style of formatting that they perform. **nroff** formats text into monospaced font, like that on an ordinary typewriter, Its output is suitable for display on the screen. **troff** formats text into proportionally spaced fonts, like those seen on this page. Its output is suitable for printing on a laser printer or other sophisticated typesetting device. The commands for **nroff** and **troff** closely resemble each other. The following descriptions will assume that you are using **nroff**, but they apply to **troff** as well.

nroff's programming language is quite complex and sophisticated. This manual includes a tutorial that introduces **nroff**'s language. You can, however, use **nroff** to perform simple formatting tasks by using the **ms** macro package. The following describes some of the more commonly used **nroff** commands.

To see how **nroff** works, type the following script:

```
cat >script.r
.ds CF "Print on Bottom of Each Page"
Here is some text.
Here is some more text.
.PP
The above command set a new paragraph.
Yet more text.
.SH
Here is a Section Heading
.PP
More text.
\fBThis is printed in bold face.\fR
This printed in Roman.
\fIThis is printed in italics or underlined.\fR
.PP
Here’s some more text.
Here’s yet more text.
And more text yet.
<ctrl-D>
```

Now, format and display the text with the following command:

```
nroff -ms script.r | more
```

You will see the text formatted for your screen. The string **Print on Bottom of Each Page** appears at the bottom of the display. The following describes the **nroff** commands with which this formatting was performed.

nroff's commands are introduced in either of two ways: by a period '.' in the *first* column of a line; or by a backslash '\' occurring anywhere in a line. The following reviews this script in detail.

.ds CF This defines the text to appear on the bottom of each page. If the text is more than one word long, it must be enclosed within quotation marks.

.PP Begin a new paragraph. **nroff** skips one line and indents the following line by five spaces (one-half inch).

.SH Print a section heading. **nroff** skips one line and prints in boldface the line of text that follows this command.

\fB Print the following text in **boldface**.

\fR Print the following text in Roman.

\fI Print the following text in *italics*.

With these few commands, you can perform simple formatting of your text.

To print the formatted text, use the command **lp**. For example, to print **script.r** on a line printer, use the command:

```
nroff -ms script.r | lp
```

This discussion is sufficient to get you started, but it just scratches the surface of what you can do with **nroff** and **troff**. See their respective entries in the Lexicon for details of what these commands can do. See the tutorial for **nroff** that appears later in this manual for a thorough introduction to the formatting language used by these commands.

Miscellaneous Commands

COHERENT includes numerous commands that perform miscellaneous tasks. These include some of the most useful, and entertaining, commands in the COHERENT system.

who: Who Is on the System

To find who is logged into the system, use the COHERENT command **who**. This command lists who is logged into the COHERENT system, one name per line. You will see your own user name there as well.

If you sit down at a terminal that is not in use, but at which someone has already logged in, the following command tells you who is logged in:

```
who am i
```

COHERENT replies with the name of the user logged in at that terminal.

write: Electronic Dialogue

The command **write** lets you carry on a “conversation” with another user. The conversation continues until you or the other user type **<ctrl-D>** on his terminal.

For example, user **fred** can begin a conversation with user **anne** by typing:

```
write anne
```

On **anne**’s terminal, the message

```
Message from fred...
```

will appear. To establish the other half of the communication, **anne** should then say

```
write fred
```

and a similar notification appears on **fred**’s terminal.

At this point, both users simply type lines on their terminal and **write** sends the message to the other user. To avoid typing at the same time, each user should end a “speech” by typing a line that has the single letter

```
o
```

to signify “over”, or “go ahead”. When the other user sends you this, you know it is your turn to “talk”, and vice versa.

When your communication is finished, you should type

```
oo
<ctrl-D>
```

Here, **oo** means “over and out”, and the **<ctrl-D>** terminates the **write** command. Note that **o** and **oo** are polite conventions, and are not necessary to using **write**.

mail: Send an Electronic Letter

You can send electronic mail to another user on your COHERENT system by using the command **mail**. This command works whether or not that person is logged into the system at the time you type your message. The message is stored in an electronic “mailbox”, and the user will be notified that a message is waiting for him the next time he logs into your system.

Before you can use **mail** on your system, you must run the program **uuinstall**. This program will ask you some questions about how you have configured your COHERENT system, and will write files of information that **mail** and the communications protocol UUCP need to deliver your mail. For detailed directions on how to run **uuinstall**, see the section *Installing UUCP* in the UUCP tutorial that appears later in this manual.

Among other things, this program will ask you to name your “site” and your “domain”. Without going into too much detail at this point, the site is *nom de plume* by which your machine is known to other COHERENT or UNIX systems. Site names generally are not computer-ese; **conan**, **terminator**, **lepanto**, **chelm**, and **smiles** are all examples of site names. If you don’t intend to communicate with other systems, use your first name as the site name. The domain is the name by which a group of related machines are together known. If you and a number of other local COHERENT systems wish to be known together, you can establish a domain and register it with the network. Domain names, too, should be descriptive. If you don’t intend to use a domain, set the domain name to **UUCP**.

To mail a message to user **anne**, just type:

```
mail anne
```

mail immediately prompts you for a title for your message:

```
Subject:
```

You can type the message’s subject, which will be used to title the message, or you can just press (¢).

Once you have titled your message, type the body of the message. You can conclude your message in any of three ways: you can type **<ctrl-D>**, type a period ‘.’ at the beginning of a line, or a question mark ‘?’ at the beginning of a line. The first two methods end the message immediately; the last method, however, invokes an editor, and lets you edit the message further before sending it on to the intended recipient. Environmental variable **EDITOR**, if defined, selects the editor to be used.

For example, to send your message to user **anne**, you might do the following. First, invoke **mail**:

```
mail anne
```

Next, give your message a title:

```
Subject:  I’ll be working late
```

Finally, type the body of the message:

```
I’ll be working late.  I hope to get home before Catherine
and George go to bed.  Please remind Ivan and Marian to do
their homework.  Marian should remember to practice her
violin.
<ctrl-D>
```

If you wish, you can first type your message into a file and then mail it. For example:

```
cat >hb.msg
All come to the birthday party at four
next to the pump room.
<ctrl-D>
```

To mail the message to user **jill**, type:

```
mail jill <hb.msg
```

You can send a mail message to several users at one time by listing each user’s name on the command line. For example, the command

```
mail jill jack ted barb < hb.party
```

mails the contents of file **hb.party** to **jill**, **jack**, **ted**, and **barb**. To illustrate the use of the mail command, send yourself a mail message. Type the following; substitute your user name for “you” in the mail command:

```
mail you
Subject:  test the COHERENT mail system
This is a note to
myself to test
mail.
.
```

If someone has sent you mail, the COHERENT system will tell you:

```
You have mail.
```

when you log in.

To receive mail, type the **mail** command with no parameters:

```
mail
```

If you have no mail, COHERENT will tell you:

```
No mail.
```

If you do have mail, the system will print each message on your terminal, along with the user name of the sender, and the date and time that the message was mailed.

After each message, the **mail** program types a question mark **?** and waits for your reply. You can type any of the following commands in reply to the prompt:

d Delete the message.

<Return>
Proceed to the next message.

s *file* Save, or copy, the message into *file*.

q Quit — exit from **mail** and return to the shell.

You will know that you are finished with all of your messages when **mail** sends you a **?** without typing anything before it.

mail can also send messages to other COHERENT or UNIX systems via the UUCP utility. See the accompanying tutorial on UUCP to see how you can set up COHERENT to do this.

msgs: Cumulative Message Board

The message of the day disappears when a new message is inserted. If a user does not log in for several days, the message of the day may no longer be there. For items that you want everyone to see, such as hours of operation or new operating procedures, you should use **msgs** instead of **motd**.

msgs helps users get all important messages, even if they don't log in every day. The system remembers which users have seen each message. After a user logs in, invoking **msgs** will show the number, date, and author of each message written since the user last logged in. Therefore it is easy for the user to stay up to date with the system-wide messages.

To add a message to the file, simply mail the message to **msgs**. To title the message, write it as the first line in the message, after the "Subject:" prompt from **mail**.

The home directory for **msgs** will grow over time, as more and more messages accumulate. Also, if a new user is enrolled on your COHERENT system, he may have to wade through several hundred messages when he first logs in. Therefore, you should purge the home directory for **msgs** every now and again; you may wish to throw away the announcements of office parties three Christmases ago, and save important information on diskette.

msgs keeps track of what messages each user has read by recording the number of the last message read in the file **$HOME/.msgsrc**. When each user logs on, his version of **.msgsrc** is inspected to determine the last message seen. If messages were added after that, **msgs** prints the ones the user wants to see, and then updates **.msgsrc**.

grep: Find Patterns in Text Files

The command **grep** lets you find lines that contain a *pattern* within one or more files. Patterns are sometimes called *regular expressions*.

To illustrate **grep**, create file **doc1** by typing:

```
cat >doc1
a few lines
of text.
<ctrl-D>
```

Then the command

```
grep text doc1
```

prints the second line of file **doc1**:

```
of text.
```

The first parameter to **grep** is the *pattern* for which you are looking; the rest of the arguments are the names of files to be examined. **text** is the pattern and **doc1** is the file.

To find if a particular user is on the system, pipe **who** into **grep**:

```
who | grep you
```

(Substitute the user name in question for **you**.) Try it with your user name. The pattern is **you**, but no file name is specified. **grep** reads input from the standard input, which in this example is connected to the output of the **who** command.

You can specify several files to be searched; simply put the additional file names after the first:

```
grep pattern doc1 doc2
```

Or, you can search all files in the current directory for the pattern with

```
grep pattern *
```

The asterisk will be interpreted to mean all files, and grep will look for *pattern* in each.

The search pattern can be a *pattern*. Patterns are fully discussed in the tutorial for **ed**.

You can also locate lines that do *not* contain given patterns by using the **grep** option **-v**.

```
grep -v bugs prog1 prog2
```

This command finds and prints all lines in files **prog1** and **prog2** that do not contain the pattern **bugs**.

date: Print the Date

The COHERENT system keeps track of the time and date. To find the date and time, use the command:

```
date
```

COHERENT responds with the day of the week, the month day and year, and the time of day.

Internally, the COHERENT system records the date and time as the number of seconds since January 1, 1970, 00:00:00 Greenwich Mean Time (GMT). This means that files created in one time zone and referenced in another time zone will bear the correct time. The time and date printed out is converted from the internal form to the local time.

passwd: Change Your Password

You should change your password from time to time, to ensure that no unauthorized person can gain access to your files (or to the system as a whole).

It is easy to change passwords on the COHERENT system: just type the command **passwd**. **passwd** first asks you for your current password (if you have one), and then asks you to enter your new password twice. Entering the new password twice helps ensure that the system gets the password as you want it. If you do not type it the same way both times, COHERENT will say:

```
Password not changed.
```

You must then begin again with the command **passwd**.

Be sure the password is something that you can remember. It is recommended that the password be at least six characters long. Do not write it down, but memorize it. You can use a four-letter password, but if you do, you should mix upper-case and lower-case letters to make it more difficult for outsiders to guess.

stty: Change Terminal Behavior

Because a wide variety of terminals can be used with the COHERENT system, you must pass some information to the COHERENT system so it can handle your terminal correctly.

The command **stty** describes the information COHERENT currently has for you; you can then use **stty** with arguments to change how COHERENT handles your terminal.

For example, COHERENT normally echoes each character you type, as you type it. However, if your terminal also echoes what you type, you will see double characters. To prevent this, issue the command:

```
stty -echo
```

The program **login** uses this feature when you type your password, to help keep it secret from anyone who is kibbitzing at your desk.

To set the echo feature again, type:

```
stty echo
```

When you first log in, the system presumes that your terminal does not directly handle the **tab** character, so when COHERENT sends a **tab** to your terminal it simulates it with spaces. If your terminal does handle tabs, issue the command:

```
stty tabs
```

The COHERENT system will no longer substitute spaces for tabs. To go back to substitution,

```
stty -tabs
```

The **<erase>** character lets you delete the previously typed character. The **<kill>** character lets you delete the line that you have been typing but have not yet finished. By default, COHERENT sets these to, respectively, **<ctrl-H>** and **<ctrl-U>**. To change them to, respectively, **<ctrl-E>** and **<ctrl-K>**, use the **stty** command as follows:

```
stty erase ^E kill ^K
```

The carat '^' tells **stty** that you want to specify a control character.

To reset erase and kill to the default values at login, the command

```
stty ek
```

suffices. This is equivalent to:

```
stty erase ^H kill ^U
```

To see what your current terminal parameter settings are, type

```
stty
```

with no arguments.

Scheduling Commands For Regular Execution

The command **cron** is a valuable tool for using your COHERENT system. With it, you can instruct COHERENT to execute commands at various times of the day or night, even if nobody is logged into the system.

To specify a command to be executed at some later time, simply enter one line of information in a "cron" file. (Where the cron file lives will described below.) For example, assume that you want to greet user **norm**, if he is logged into the system on Monday morning. You can do this by sending him a message at 8:13 on Monday. Use MicroEMACS to add the following line to the cron file:

```
13 8 * * 1     msg norm%You are sure in early!
```

The numbers and * at the beginning specify the time:

```
13 8 * * 1
```

The **13** means "13 minutes past the hour". (**cron** numbers the minutes zero through 59.) The **8** means "8 AM". (**cron** numbers the hours of the day zero through 23, with zero indicating 12 AM.) The positions containing * normally specify the day and month. The two * characters mean "any day" and "any month". Finally, the **1** means "day 1 of the week," which is Monday. (**cron** numbers the days of the week zero through six, with zero indicating Sunday.) The breakdown of this command is shown as follows:

minute	13
hour	8
day of month	* — all days
month	* — all months
day of week	1 — Monday

Because each entry in the cron file must be on one line, the symbol **%** represents the beginning of the input string. If the information is too long for one line, type a backslash character before you press (¢) to end the line. The backslash tells **cron** to ignore the **<Return>** character.

With this information in the file, **cron** executes the command

```
msg norm
Am Monday!
```

at 8:13 every Monday morning.

cron expects time to be in the 24-hour clock, so 1 PM is represented as **13** hours. If you need to print a literal percent sign '%', precede it with a backslash:

```
\%
```

The times for **cron** commands can be even more complex than the numbers and * shown above.

You can express a range for any of the five parts of a time by separating two numbers with a hyphen. For example, to send user **marianne** a humorous message on week days, use the command:

```
59 11 * * 1-5   /usr/games/fortune | msg marianne
```

To list a choice of times, separate single numbers or ranges with commas but no spaces. To send notification about a meeting on Monday, Wednesday, and Friday at 3 PM, use:

```
0 15 * * 1,3,5  echo Meeting at 3:30 ... | mail fred anne joe
```

The time specification

```
0 15 * * 1,3,5
```

represents the time 1500 (3 PM) on every Monday, Wednesday, and Friday.

mail and **msg** are just some examples of commands that can be used with **cron**; many others can be used. For example, **cron** is commonly used to execute UUCP commands late at night, when telephone rates are low. See the Lexicon article on **cron** for more information about this command. If you wish to schedule commands to be run but not on a regular basis, use command **at**. See its Lexicon article for further details.

As was mentioned above, you must edit a cron file for the **cron** daemon to execute a command automatically. COHERENT uses a complex scheme of cron files. If the file **/usr/lib/crontab** exists, the **cron** daemon will read it — and only it. However, if **/usr/lib/crontab** does not exist, the **cron** daemon will look in directory **/usr/spool/cron/crontabs** for the set of cron files located there. Each user can have his own cron file. All commands in a user's cron file are executed with the suite of permissions granted to that user; thus, a user cannot use **cron** to delete files that do not belong to him. If a user wishes to create or update his own cron file, use the command **crontab**; see its entry in the Lexicon for details.

Please note that each flavor of cron file uses the syntax described above.

Managing Processes

A *process* is a command that is undergoing execution. Because COHERENT is a multi-tasking operating system, numerous processes can be undergoing execution at the same time. The following commands let you monitor and, within limits, affect the operation of the processes your COHERENT system is executing.

ps: List Active Processes

Each process in the system is assigned a number called the *process id*, or *PID*. Each user logged into the system has one or more processes. Except in special circumstances, the first process that he has is the shell, or command-line interpreter. The commands he types are run by the shell.

The shell normally waits for a command to terminate before it begins to process the next command. However, if you use the '&' operator, the shell creates simultaneous processes: that is, while it executes one command it will let you type another. Thus, you can execute two or more commands simultaneously.

You can examine the processes associated with your login, or all processes in the system, with the command **ps**. Type:

```
ps
```

The result will resemble:

```
TTY       PID  COMMAND
color0     36  ksh
color0   4004  mail
color0   4005  me
color0   4009  sh
color0   4010  ps
```

The first column names the terminal you are running on, in this case the virtual console **color0**. This identifier is taken from the file **/etc/ttys**, with the prefix **tty** removed from name. The **tty** identifier is also printed by the command **who**. The second column lists the corresponding process identifier (PID). The third column names each command and gives its parameters, if any. **ksh** represents the shell process, and **ps** represents the **ps** command itself.

To see all the processes, type:

```
ps -a
```

The result will resemble:

```
TTY       PID  COMMAND
null        1  init
color3     33  ksh
color2     34  ksh
color1     35  ksh
color0     36  ksh
com3l      37  login
color3   3629  sh
color3   3630  kermit
color3   3631  kermit
color0   4004  mail
color0   4005  me
color0   4011  sh
color0   4012  ps
```

This display will, of course, differ quite a bit from system to system and from minute to minute.

For a full description of all options to **ps**, see its entry in the Lexicon.

kill: Signal Processes

Occasions will arise when the system administrator must log other users out of the system. For example, you may need to bring the system down quickly; or perhaps a user forgot to log out before leaving the terminal and did not see your broadcast message requesting that all users log out.

The command **kill**, when used by the superuser, terminates processes. To log out a user whose shell has process number 300, use the command:

```
kill -9 300
```

You must be logged in as **root** or use the command **su** to **kill** a process that belongs to another user. Each user can kill all processes that he owns, including his own shell process (which automatically logs him out).

kill has other uses as well — see the Lexicon's entry for **kill** for more information.

Programming Under COHERENT

The COHERENT system provides a number of languages in which you can write programs.

The shells included with COHERENT — **sh**, the Bourne shell, and **ksh**, the Korn shell — not only process commands, but are powerful programming languages in their own right. For details on how to program in these languages, see their respective entries in the Lexicon; and see the tutorial *Introducing sh, the Bourne Shell*, which follows in this manual.

COHERENT includes a full-featured assembler, with which you can assemble your assembly-language programs. Assembly language is sometimes required for operations that require you to work very closely with the operating system or hardware. For more information on the COHERENT assembler, see the Lexicon entry for **as**.

Most programming that cannot be executed efficiently by a shell language is done in C, the language in which the COHERENT system was written. The COHERENT system comes with a full-featured C compiler, with which you can compile the program you write in that language. If you are new to C, the tutorial *The C Language*, which follows in this manual, will introduce you to it. The following sub-sections briefly describe the tools available under COHERENT with which you can write, compile, and debug your C programs.

Basic Steps in COHERENT Programming

The steps that are necessary to generate a program are:

1. Create the program source file
2. Compile the source program, correcting any errors
3. Test and debug the program
4. Run the program

If you have compilation errors in step 2, or program errors in step 3 or 4, return to step 1.

Use your favorite editor to build and change the source program, the **cc** command to compile the source program and produce an object program, and **db** to help debug the program. Although the C compiler provides a macro facility, other languages do not. Therefore, if the source program uses macros, you can use **m4** to expand the macros.

This section covers each of these steps and provides some example programs.

Create the Program Source

The first step is to use MicroEMACS, **vi**, **ed**, or some other editor to create the program's source file. Details on the use of **ed** and MicroEMACS are covered in their respective tutorials, which follow in this manual. Each editor's commands are summarized in its Lexicon article.

For the first program, try a simple program that prints a short message on your terminal. For the sake of simplicity, we'll enter text using **cat** instead of invoking an editor. To build the program, type the following:

```
cat > small.c
main ()
{
        printf ("The COHERENT operating system\n");
}
<ctrl-D>
```

The first line invokes the concatenation program **cat** to enter the program's source code. The **<ctrl-D>** signals that you have finished entering text.

The program itself begins with the special word **main** which defines a function and must appear in every C program. The parentheses, here with nothing between them, enclose any arguments that are passed to the function. They are required even if there are no arguments. The body of the program appears between the braces **{** and **}**.

The function **printf** is part of the standard library of C programs. It prints formatted information on the terminal. In this case it will produce the string enclosed between quotation marks. The special character string

```
\n
```

means "newline". Two lines of output to the terminal can be produced by

```
"line 1\nline 2\n"
```

as an argument to **printf**. This appears in the output as:

```
line 1
line 2
```

For a fuller introduction to the C language, see the tutorial on the *The C Language*, which follows in this manual.

cc: Compile the Program

The command **cc** compiles C programs. It executes all the parts of the C compiler and the associated linker **ld**. The linker combines pieces of programs and includes necessary elements from the library, such as **printf()**. The linker is occasionally called from the command line, but only for more complex problems than you are trying here. To compile our test program, type the command

```
cc small.c
```

If the compiler detects any errors, it prints a message on the terminal, along with the line number that contains the error. You can use this line number to find the error with your editor and fix it. You can now use the program by simply typing:

```
small
```

The tutorial on *The C Language* describes **cc** in greater detail; also see its entry in the Lexicon for a full summary of its many capabilities.

m4: Macro Processing

To extend the capabilities of all languages, the COHERENT system provides a macro processor, called **m4**.

Program source for all languages consists of character strings. Macro processors perform string replacement, whereby a string in the input file may be replaced by another string. **m4** provides parameter substitution, as well as testing values of currently available strings and conditional processing. **m4** is unique in that you can rearrange large sections of the input text by using the macros. For more information on **m4**, see the tutorial *Introduction to the m4 Macro Processor*, which follows in this manual.

make: Build Larger Programs

All the examples of programs thus far have been self-contained. As programs grow larger, it is usual to divide the source program into smaller files. This simplifies editing, speeds compilation, increases modularity, and lets several different programs share common functions.

Thus, in developing the larger program, you may have several source files in your directory, possibly a header file or two, and the object files that result from compilation. From these are built the executable file that runs when you type its name.

To change or fix the program, you must edit the source programs or header files in question with **ed**, recompile the required source, and relink all the modules. But, with a change that affects several modules, it can be tricky to remember exactly which modules need recompilation, and it can be time-consuming to recompile all modules.

COHERENT provides a command called **make**, which solves this problem. **make** examines the time a file was last modified, and the time of modification of files that it depends upon, and performs the necessary compilation or other processing. (COHERENT file system directories contain the time that each file was created or modified.)

The tutorial *The make Programming Discipline*, which follows in this manual, fully introduces this powerful and useful program.

db: Debug the Program

The first and most critical step to debugging programs is to not put bugs in them! The methods of structured analysis, design, and programming, or the method of stepwise refinement can substantially reduce the number of errors in a program.

One can also place **printf** statements at strategic points throughout the program to display logic flow and key data values. These display statements should be designed so that they can be turned off for normal operation without removing them from the program.

On occasion, however, you may find that it is necessary to debug at the machine level. If you must, COHERENT's **db** will make it possible to do so.

db provides tools that make the machine program instructions visible in the most natural notation. That is, instructions are displayed in a fashion that resembles assembly language, numbers can be displayed in hexadecimal, octal, or decimal as needed, and strings of characters displayed in familiar graphic form. **db** can also patch a program to be run again, as well as to control the execution of a program with breakpoints and one step at a time.

Briefly, to use **db** on a program like our sample **small** above, use the command:

```
db small
```

Now you can inspect and display instructions and data in the system, control execution, and even change the instructions in the program if you are bold enough.

To examine a data segment location in the program, simply type the address of the location. **db** knows about symbols in the program, so if you want to examine the location corresponding to **main**, type:

```
main
```

db types out the value in hexadecimal or octal (depending upon which is appropriate for your machine).

You can expand the display command to print many locations at one time, and choose the format of printout. To print five locations interpreted as instructions, type

```
main,5?i
```

where the format character **i** follows the question mark indicating format, and 5 is the count of locations to be printed. To exit **db**, type

```
:q
```

For a complete list of the format that **db** recognizes, and other details about **db**, see its entry in the Lexicon.

Administering the COHERENT System

The COHERENT system can be used by many people at the same time. One person must coordinate its use, like a key operator does for an office copier. This person is called the *system administrator*, and he sees to it that the COHERENT system runs smoothly every day. The administrator can also customize the COHERENT system to the needs of an individual installation.

Although you may be the only person to use your COHERENT system, many of the ideas discussed here are important for making your system work at its best. Please spend a few minutes reading this manual to familiarize yourself with the elementary concepts of COHERENT system maintenance.

Adding a New User

Each user allowed to use your COHERENT system must have a *user name* and a *user id*; the user may also have a *password*. The user name is usually the user's initials or a nickname. The *user id* is an integer number used to identify the user internally to the system. As system administrator, you will assign both of these for each user. This section tells you how.

To log in to the system, a user must have an entry in the *password* file **/etc/passwd**. The password file contains each user's name, id, and password if any. As system administrator, you will maintain this file.

Likewise, each group of users is assigned a *group name*, as well as a *group id*. Groups are not necessary to use the COHERENT system, but some installations prefer to set up groups by project or department.

It is simple to add a new user to the system. The command **newusr** takes care of all the details, and makes an entry in the password file. You must be logged in as **root**. For example, to create an entry for a user named Henry, log in as **root**, and then issue the command:

```
/etc/newusr henry "Henry Smith" /usr
```

This creates an entry in **/etc/passwd** for **henry**, creates his home directory in the **/usr** file system, creates all appropriate files for him (such as his **.profile** and his mailbox), and sets all permissions correctly.

System Security

One of the most important tasks in running your COHERENT system is maintaining its security. Basically, security means two things: keeping outsiders from logging into your system, and keeping your system's users from doing untoward things. This section describes some steps you can take to ensure that your system is secure.

Passwords

Passwords provide the first level of COHERENT system security.

For systems with passwords, each user with a password must type his password as part of the login process. If he enters the password incorrectly, he cannot log in.

Your system's administrator can assign a password when she creates a user's log-in account, as described above. If you do not assign a password, anyone will be able to log in as that user.

In any system with passwords, it is especially important to assign a password to the **root**, or *superuser*. If the superuser does not require a password, any user can log in as **root** and automatically have access to the powerful tools that control the operation of the system.

Any user with a password can restrict access to his files. Once you assign him his password, he can change it with the command **passwd**. However, because of higher privileges, **root** can always access everyone's files.

The passwords are kept in file **/etc/passwd**, with the rest of the user login information. Passwords are encrypted, so reading **/etc/passwd** will not reveal passwords.

File Protection

The second level of COHERENT system security is *file protection*. A user can set each of three categories of protection for each of his files. A standard protection, or *access permission*, is given to each file when it is created.

The three categories of permissions are for the user himself, for other users in his group, and for all other users. To see the levels of protection of your files, type the command

```
ls -l
```

For more details on the meaning of each column in this printout, see the Lexicon entry for the change-mode command **chmod**.

Encryption

The command **crypt** provides a third level of system security. It lets a user encode and decode information in a file. The superuser has access to every file in the system; so to protect sensitive information even from his prying eyes, a user can disguise it with encryption. Sensitive system information, such as passwords, are also encrypted for security purposes; and the **mail** command lets users send encrypted mail to each other. For details about encryption, see the entry on **crypt** in the Lexicon.

Dumping and Saving Files

You should regularly copy your files on floppy disk, to protect your valuable files against a catastrophic system failure. The Lexicon article **backups** describes in detail how to do this.

System Accounting

The COHERENT system provides two types of computer time *accounting* to help you track the use of the system. Three commands control the accounting and provide reports at various levels of detail.

Note that system accounting adds overhead to your system, because your system has to do more work to record everything it does, and because the accounting files can quickly grow to unmanageable sizes. System accounting is useful for COHERENT systems that are being used by multiple users who must account for (i.e., pay for) their use of the system, or in other circumstances where it is important to note each user's activity. For most systems that support a handful of users, system accounting simply isn't worth the bother.

If, however, you decide that you need system accounting, read on.

ac: Login Accounting

Whenever a user logs into the COHERENT system, it records the user's name, the terminal number, and the date and time of the login. It also records when he logs out.

You can use this information to compute the time each user, or all users, were logged into the system. The command **ac** prints the total of all login times recorded in the accounting file. An example of the result is

```
Total:  8357:00
```

You can ask for a summary of total login times for each day by typing:

```
ac -d
```

An example result would be:

```
Friday November 13:
        Total:  53:08
Saturday November 14:
        Total:  75:36
Sunday November 15:
        Total:  73:15
```

Finally, you can summarize the times for individual users with the command:

```
ac -p jack ted fred
```

This will show the total login times for these users:

```
fred            1100:42
jack            910:41
ted             641:58
Total:          2653:21
```

Also,

```
ac -pd
```

gives the time for each user, for each day that he logged in.

Login accounting is not automatically operational. The login information is collected only if the file **/usr/adm/wtmp** exists.

To start login accounting if it is not working, type the command

```
>/usr/adm/wtmp
```

while logged in as **root**. This creates the file **/usr/adm/wtmp** if it does not exist (and destroys existing information if it does) and thereby enables login accounting.

To turn off login accounting while it is running, you can type:

```
rm /usr/adm/wtmp
```

After you activate login accounting, you should purge **/usr/adm/wtmp** periodically as it grows continuously, and on an active system will eventually consume much disk space. To purge the current information but leave accounting turned on, type:

```
>/usr/adm/wtmp
```

sa: Processing Accounting

While login accounting tells you how much time a user spends logged into the system, it does not tell you the individual commands used. *Process accounting* does so. Under COHERENT, each execution of each command constitutes a separate process. (COHERENT's ability to maintain a list of processes and swap each in and out of memory until all are executed, is what gives COHERENT its multi-tasking capability.) Process accounting records system time, user time, and real time for each command executed by each user on the system. The command **sa** reports this information for you, using a format that you set.

sa has several options, to generate different reports. When used with no options, **sa** lists the number of times each call is made, the total CPU time, and the total real time used by the command, ordered by decreasing CPU time. This is a summary by command; the following gives an example:

```
           #CALL  CPU    REAL
sh         61     1      832
ld         5      1      7
ar         5      0      1
ranlib     3      0      1
p          16     0      11
dld        2      0      1
lc         19     0      1
cc         4      0      8
atrun      43     0      1
find       1      0      0
ed         1      0      2
           ...
```

This report has been truncated to save space. The listing will depend on what commands are used in your system, and the characteristics of your hardware. To summarize by user, use the **-m** option:

```
sa -m
```

The option **-l** separates CPU time expended by users from that expended by the system. This command

```
sa -l
```

produces:

```
            #CALL   USER    SYSREAL
sh          61      0       1832
ld          5       0       07
ar          5       0       01
ranlib      3       0       01
p           16      0       011
dld         2       0       01
lc          19      0       01
cc          4       0       08
atrun       43      0       01
find        1       0       00
ed          1       0       02
cat         4       0       01
rm          3       0       00
            ...
```

This report has been truncated to save space. To list the user name and the command name, use **sa** with the option **-u**. No times or counts are given. The command:

```
sa -u
```

produces output of the form:

```
tj          p
tj          lc
tj          find
tj          pr
bin         lc
tj          spin
tj          sh
bin         cc
...
```

This report has been truncated and edited to save space. In practice, it is longer. The **-u** option overrides other options.

Process accounting is on only if you turn it on. To turn on process accounting, type the command:

```
/etc/accton /usr/adm/acct
```

while logged in as **root**. The file **/usr/adm/acct** holds the raw accounting information.

To turn off process accounting, use the same command with no file name:

```
/etc/accton
```

If accounting is not on when you type this command, you will get an error message. No information is gathered when accounting is turned off.

When process accounting is in use, the file **/usr/adm/usracct** grows with each user command issued. You should regularly condense or remove the information, to keep the file from devouring all free space on your disk. To condense the information, invoke **sa** with the **-s** option. You must turn off accounting while condensing information.

The information summarized by user will appear in **/usr/adm/usracct**, and information saved by command is placed in **/usr/adm/savacct**. These summarized files are used in future requests to **sa**. After condensing, you can turn accounting back on.

Additional options give flexibility to the report. See the entry for **sa** in the Lexicon for additional details on these options.

Conclusion

The following sections of this manual give tutorials to teach you how to use many of COHERENT's tools and commands. The Lexicon contains brief synopses of all commands, library routines, system calls, and macros available under the COHERENT system. It also includes many technical references and definitions, to help you with terminology throughout this manual.