4

The OS-9 File System

OS-9 File Storage

All information stored on an OS-9 computer system is organized into files and directories. Files and directories provide a way for you to organize your information. A file may contain a program, data, or text. A directory is a file containing the names and locations of the files and directories it contains. This allows you to organize your files by topic, work group, etc.

When a file is created, the information is stored as an ordered sequence of ***bytes***. These bytes are organized into ***sectors***. A sector is a pre-defined group of bytes. For example, a sector may be composed of 256 bytes. This means that every 256 bytes are grouped together as a sector.

During the format procedure, each sector is marked as being unused. The allocation map keeps track of each sector. If a sector is in use, it is marked in the allocation map located at the beginning of each disk as being in use. When a file is created, the information is stored in sectors. When a file is expanded, the new information is stored in sectors. When a file is shortened or deleted, the previously used sectors are unmarked in the allocation map and are available for use by other files.

Within a text file, each byte contains one character. Data is written to a file in the order it is provided. Data is read from a file exactly as it is stored in the file.

When a file is created or opened, a file pointer is also created and maintained for it. The file pointer holds the address of the next byte to be written or read (see Figure 4a). As data in the file is read or written, the file pointer is automatically moved. Therefore, successive read or write operations transfer data sequentially (see Figure 4b).

You can directly access any part of a file by positioning the file pointer to any location in the file using an OS-9 system call: seek. You can access the seek system call through the various languages available for OS-9 or directly with the macro assembler command: I$SEEK. I$SEEK is described in the ***OS-9 Technical Manual***.

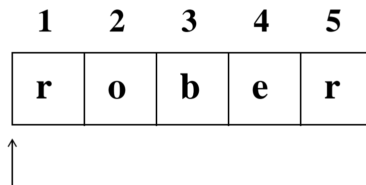


Figure 4a: When creating or opening a file, the file pointer is positioned to read from or write to the first component.

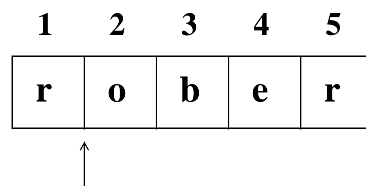


Figure 4b: After reading or writing the first component of a file, the file pointer points to the second component.

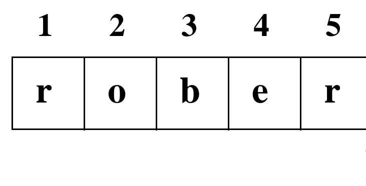


Figure 4c: The file pointer is pointing to the current end-of-file. Attempting another read operation causes an error. Another write operation increases the size of the file.

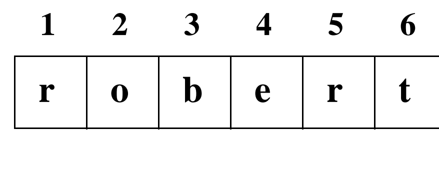


Figure 4d: The next write operation adds a new component to the file and moves the file pointer to the new end-of-file.

Reading up to the last byte of the file causes the next read operation to return an end-of-file status (see Figure 4c). Trying to read past the end-of-file mark causes an error. To expand a file, simply write past the previous end of the file (see Figure 4d).

Because all OS-9 files have the same physical organization, you can generally use file manipulation utilities on any file regardless of its logical usage. The main logical types of files used by OS-9 are:

- Text files
- Executable program module files
- Data files
- Directories

Directory files are an exception and are discussed separately.

Text Files

Text files contain variable length lines of ASCII characters. Each line is terminated by a carriage return (hex $OD). Text files typically contain documentation, procedure files, program source code, etc. You can create text files with any text editor or the build utility.

Executable Program Module Files

Executable program modules store programs generated by assemblers and compilers. Each file may contain one or more modules with standard OS-9 module format. See the ***OS-9 Technical Manual*** for more information on modules.

Random Access Data Files

A random access data file is created and used primarily by high level languages such as C, Pascal, and BASIC. The file is organized as an ordered sequence of records of varying sizes. If each record has exactly the same length, its beginning address within the file can be computed to allow records to be accessed in any order. OS-9 does not directly deal with records other than providing the basic file manipulation functions high level languages that support random access records require.

File Ownership

When you create a file or directory, a ***group.user ID*** is automatically stored with it. The group.user ID is formed from your group number and your user number. The group number allows people that work on the same project or work in the same department to share a common group identification. The user number identifies a specific user. Therefore, a group.user ID identifies a specific user in a specific group or department.

The group.user ID determines file ownership. OS-9 users are divided into two classes:

- The ***owner***
- The ***public***

The owner is any user with the same group or user number as the person who created the file. This means that any user with the same group number as the person who created the file can access the file in the same way as the creator of the file. Likewise, any user with the same user number is considered the owner.

The public is any person with a group.user ID that differs from the person who created the file.

> + A user with a group.user ID of 0.0 is referred to as a ***super user***. A super user can access and manipulate any file or directory on the system regardless of the file's ownership.

On multi-user systems, the system manager generally assigns the group.user ID for each user. This number is stored in a special file called a password file. A super user on a multi-user system is generally the system manager, although other people such as group managers or project leaders may also be super users.

NOTE: Password files are discussed in the chapter on the shell.

On single-user systems, users have super user status by default.

Attributes and the File Security System

File use and security are based on file attributes. Each file has eight attributes. These attributes are displayed in an eight character listing.

The term ***permission*** is used when one of the eight possible attribute characters is set. Permission determines who can access a file or directory and how it can be used. If a permission is not valid for the file or directory being examined, a hyphen (**-**) is in its position.

Here is an attribute listing for a directory in which all permissions are valid:

dsewrewr

By convention, attributes are read from right to left. They are:

Attribute	Abbreviation	Description
Owner Read	r	The owner can read the file. When off, this denies any access to the file.
Owner Write	w	The owner can write to the file. When off, this attribute can be used to protect files from accidentally being deleted or modified.
Owner Execute	e	The owner can execute the file.
Public Read	pr	The public can read the file.
Public Write	pw	The public can write the file.
Public Execute	pe	The public can execute the file.
Single user	s	When set, only one user at a time can open the file.
Directory	d	When set, indicates a directory.

The OS-9 File System

OS-9 uses a ***tree-structured***, or hierarchical, organization for its file system on mass storage devices such as disk systems (see Figure 4e). Each mass storage device has a master directory called the ***root directory***.

The root directory is created automatically when a new disk is formatted. It contains the names of the files and the sub-directories on the disk. Every file is listed in a directory by name, and each file has a unique name within a directory.

An OS-9 directory can contain both files and sub-directories. Each sub-directory can contain more files and sub-directories. This allows subdirectories to be imbedded within other subdirectories. The only limit to this division is the amount of available disk space.

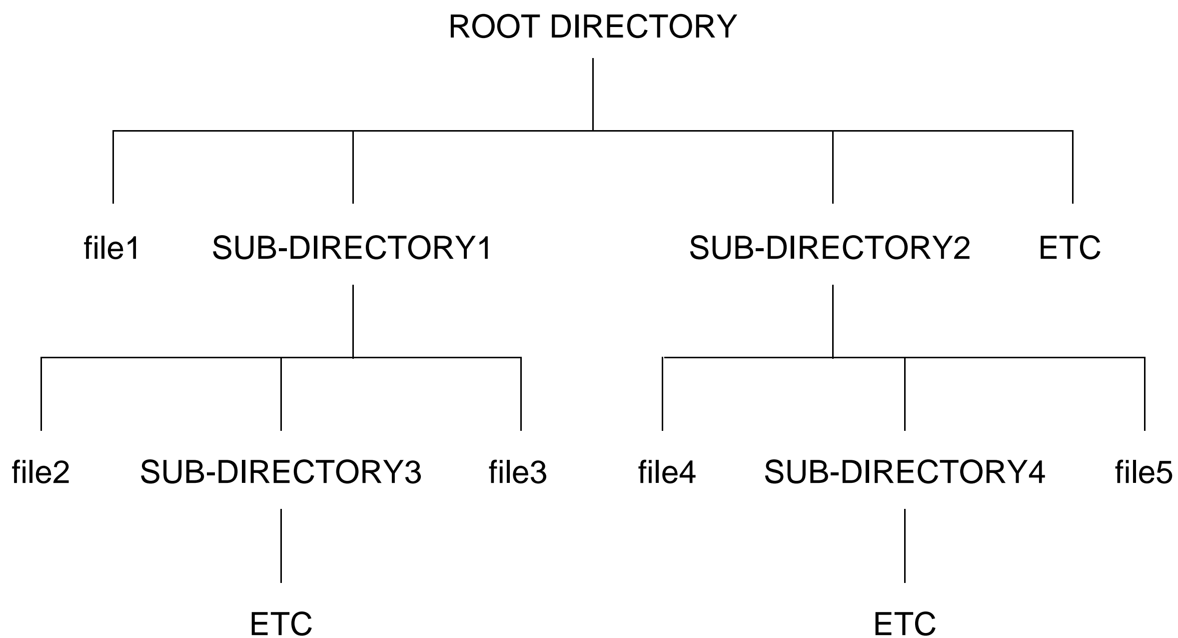


Figure 4e: The File System

With the exception of the root directory, each file and directory in the system has a ***parent*** directory. A parent directory is the directory directly above the file or directory being discussed. For example in Figure 4e, the parent directory of file2 is SUB-DIRECTORY1. Likewise, the parent directory of SUB-DIRECTORY1 is the root directory.

Current Directories

Two working directories are always associated with each user or process. These directories are called the ***current data directory*** and the ***current execution directory***.

> A ***data directory*** is where you create and store your text files.
>
> An ***execution directory*** is where executable files such as utilities and programs you have created are located.

The current directory concept allows you to organize your files while keeping them separate from other users on the system. The word ***current*** is used because you can use the chd command to move through the tree structure of the OS-9 file system to a different directory. This new directory then becomes your current data or execution directory.

NOTE: The chd utility is discussed later in this chapter.

On a single user system, OS-9 chooses the root directory of your system disk as your initial current data directory. Your initial current execution directory is the CMDS directory. The CMDS directory is located in the root directory of the system disk.

On a multi-user system, your current data and execution directories are established for you as part of the initial login sequence. When you login, your initial directories are set up according to your password file entry. A password entry is established for each user on a multi-user system. This entry lists the user's password, current directories, etc. For more information on password files, see the chapter in this manual on the shell and the login utility in the ***OS-9 Utilities*** section.

Your execution directory on a multi-user system is usually the CMDS directory. The CMDS directory is shared with other users. CMDS contains OS-9 utilities and other executable files. If all users had their own copy of all OS-9 commands, a great deal of disk space would be wasted. Private execution directories are also possible and are discussed later in this chapter.

The Home Directory

On typical multi-user systems, all users have their own data directory, but share an execution directory. The private data directory allows you to organize your own files by project, function, or any other method without affecting other user's files. The data directory specified in the password file entry is known as your ***home directory***. When you first login to the system, you are placed in this directory. Using the chd utility with no parameters also places you in this directory. The chd utility is discussed later in this chapter.

On single user systems, you may establish a home directory by setting the HOME environment variable. Refer to the chapter on the shell for more information on setting the HOME environment variable.

Directory Characteristics

Some important characteristics relating to directory files are:

- Directories have the same ownership and attributes as regular files. However, directories always have the d attribute set.
- Each file name within a directory must be unique. For example, you cannot store two files with the name of trial in the same directory. Files can have identical names, as long as they are stored in different directories.
- All files are stored on the same device as the directory in which they are listed.
- The only limit to the number of files that can be stored in a directory is the amount of free disk space.

Accessing Files and Directories: The Pathlist

You can access all files or directories in your current data directory by specifying the name of the file or directory after the proper command. When only a file or directory name is given, OS-9 will not look outside your current data directory to find it.

If you want to access a file that is not in your current data directory or run a program that is not in your current execution directory, you must either change your current directory or specify a ***pathlist*** through the file system for OS-9 to follow.

There are two types of pathlists:

- Full pathlists
- Relative pathlists

A full pathlist starts at the root directory and follows the directory names in the list down the file structure to a specific file or directory. A full pathlist must begin with a slash character (/). Slashes separate names within the pathlist.

The following example is a full pathlist from the root directory, /d1, through two subdirectories, PASCAL and TESTS, to the file futureval.

/d1/Pascal/tests/futureval

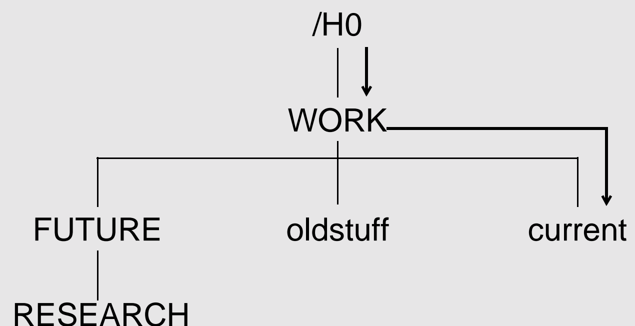
The next example specifies a path from the root directory, /h0, through the USR subdirectory to the NICHOLLE subdirectory.

/h0/usr/nicholle

Full Pathlist:

A full pathlist begins at the root directory regardless of where your current data directory is located. It lists each directory located between the root directory and a specific file or subdirectory.

Example: Your data directory is RESEARCH. A full pathlist to current is /h0/work/current.



A ***relative*** path starts at the current directory and proceeds up or down through the file structure to the specified file or directory. A relative pathlist does not begin with a slash (/). Slashes separate names within a relative pathlist.

When you use a relative pathlist and the desired destination requires going up the directory tree, you can use special naming conventions to make moving around the pathlist easier. A single period (.) refers to the current directory. Two periods (..) refer to the current directory's parent directory. Add a period for each higher directory level. For example, to specify a directory two levels above the current directory, three periods are required. Four periods refer to a directory three levels above the current directory.

Relative Pathlist:

A relative pathlist begins at your current directory regardless of its location in the overall file structure.

Example: Your data directory is RESEARCH. A relative pathlist to current is .../current .

NOTE: Using these name substitutes does not change the actual directory's name.

The following example is a relative pathlist which begins in your current directory and goes through the subdirectories DOC and LETTERS to the file jim.

doc/letters/jim

The next pathlist goes up to the next directory above your current directory and then through the subdirectory CHAP to the file page.

../chap/page

The next pathlist specifies a file within your current directory. No directories are searched other than the current directory.

accounts

Basic File System Oriented Utilities

This section explains some of the OS-9 utility commands that manipulate the file system. The utilities include dir, chd, chx, pd, build, makdir, list, copy, dsave, del, deldir, and attr. The examples given refer to the file system diagram in Figure 4f.

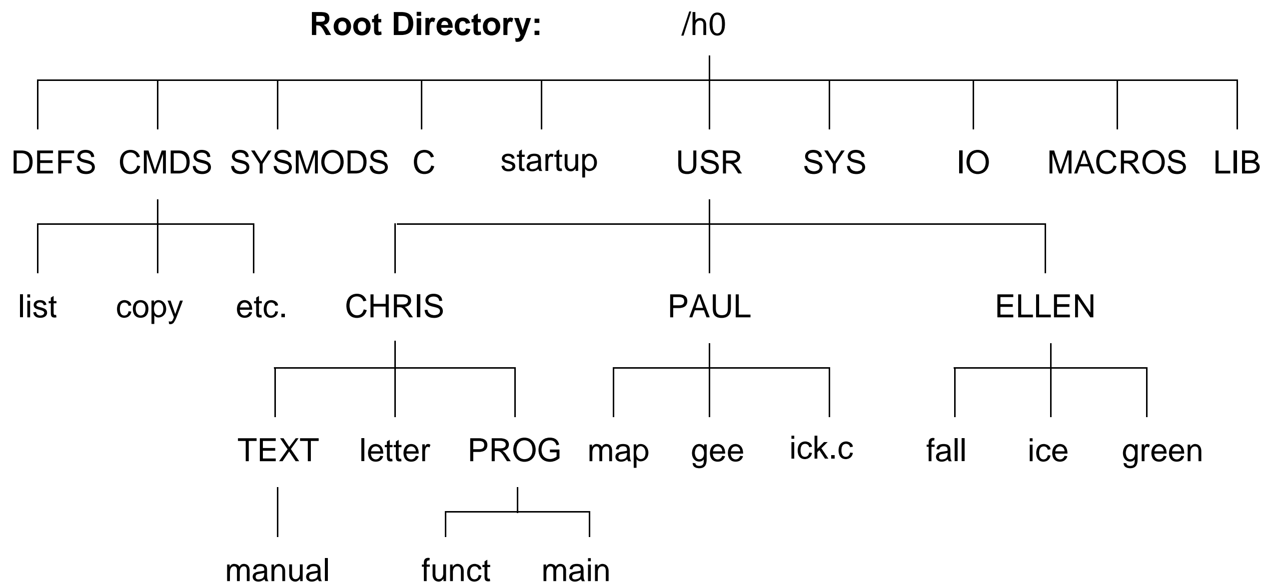


Figure 4f: Diagram of a Typical File System

Dir: Displaying the Contents of Directories

The dir utility displays the contents of directories. Typing dir by itself displays the contents of your current data directory. For the following example, the current data directory is /h0 in Figure 4f:

```
$ dir

 directory of .  13:56:58
C           CMDS        DEFS        IO          LIB
MACROS      SYS         SYSMODS     USR         startup
```

To look at directories other than your current data directory, you must either provide a pathlist to the desired directory or change your current data directory. Changing directories is discussed later in this chapter.

> To display the contents of another directory without changing your current data directory, type dir and the pathlist to the directory.

For example, if you are in the root directory and you want to see what is in the DEFS directory, type:

```
dir defs
```

dir now displays the names of the files in the DEFS directory. The name defs is a relative pathlist. You can type dir defs because DEFS is in your current data directory. You can also use the full pathlist, dir /h0/defs, and get the same result.

To display the contents of your current execution directory, type dir -x.

You may also use wildcards with the dir utility and with most other utilities as well. OS-9 recognizes two wildcards: the asterisk (*) and the question mark (?). An asterisk is replaced by any number of letter(s), number(s), or special characters. Consequently, an asterisk by itself expands to include all of the files in a given directory. A question mark is replaced by a single letter, number, or special character.

For example, the command dir * lists the contents of all directories located in the current data directory. The command dir /h0/cmds/d* lists all files and directories in the CMDS directory that begin with the letter d. The command dir prog_? lists all files in your current directory that have a file name with prog_ followed by a single character.

For more information, see the section on wildcards in the chapter on the shell.

Dir Options

dir has several options which are fully documented in the ***OS-9 Utilities*** section. Some of these options are discussed here. Try each of the options and see what information is displayed.

The -e option gives an ***extended directory listing***. An extended directory listing displays all files within the specified directory with their attributes, the size of the file, and the sector where the file is stored. The following example uses the file structure shown in Figure 4f.

```
$ dir usr/chris -e

                              Directory of USER/CHRIS 12:30:00
 Owner    Last Modified  Attributes Sector Bytecount Name
-------   -------------  ---------- ------ --------- ----
 12.4     89/06/17 1601   ------wr    3458      5744 letter
 12.4     89/07/03 1148   d-----wr   104A0     15944 PROG
 12.4     89/05/13 1417   d-----wr    DODO     11113 TEXT
```

The -r option displays the contents of the specified directory and any files contained within its subdirectories. Using Figure 4f as an example, typing dir usr/chris -r lists the following:

Directory of . 12:30:15

PROG	**TEXT**	**letter**

Directory of PROG 12:30:15

funct	**main**

Directory of TEXT 12:30:15

manual

You can use the dir options with each other. Typing dir -er displays all files within the current data directory, all files within its sub-directories, and provides an extended listing of their attributes, sizes, etc.

Chd and Chx: Moving Around in the File System

The chd and chx utilities allow you to travel around the file system.

> ✛
> - The chd utility allows you to change your current data directory.
> - The chx utility allows you to change your current execution directory.

To change your current data directory, type chd followed by a full or relative pathlist. For example, if your current data directory is /h0 and you want your current data directory to be USR, you would type chd and the pathlist of USR.

For example, with a relative pathlist, type:

chd usr

With a full pathlist, type:

chd /h0/usr

Your current data directory is now USR. If you type dir, you will see the contents of USR:

directory of . 14:04:32

CHRIS	**ELLEN**	**PAUL**

If you want to see which files are in the CHRIS directory, type dir chris. Or change directories by typing chd chris and after the new prompt, type dir.

If you want to return to your home directory, which in this case is /h0, type chd without a pathlist. After changing directories, dir displays the contents of /h0.

The chx command allows you to redefine an existing directory as a personal execution directory. This may be important if you have programs you do not want other people to execute. To use this command, type chx, followed by a full or relative pathlist to the directory. When using a relative pathlist with chx, the pathlist is relative to your current execution directory.

If your current data directory is USR and you want to change your current execution directory from CMDS to PAUL, you could type the relative pathlist chx ../usr/paul or the full pathlist chx /h0/usr/paul. When you type a command after you have changed your current execution directory, PAUL is searched instead of CMDS.

Typing dir -x displays the contents of your current execution directory, PAUL:

```
                              directory of . 14:05:06
    gee                       ick.c                         map
```

Climbing Directory Trees

You can use OS-9's special naming conventions to move around the file system. As a reminder, the naming conventions are periods specifying the current directories and directories higher in the file structure. For example:

- **.** refers to the current directory
- **..** refers to the parent directory
- **...** refers to two directory levels higher
- etc.

When used as the first name in a path, you can use these naming conventions in conjunction with relative pathlists.

NOTE: If you are planning to port your code to other operating systems, you must remember that most operating systems only use this convention as it refers to the current and parent directories. For example, if you use ... to refer to the directory above a parent directory, most operating systems require you to use ../.. instead.

The examples below relate to the file structure in Figure 4g. The examples assume your initial current data directory is PROG.

The following example displays the contents of PROG. It is functionally the same command as dir:

```
dir .

 directory of . 14:04:32
funct                   main
```

The next command displays the contents of PROG's parent directory, CHRIS.

dir ..

directory of .. 14:05:58
PROG TEXT letter

This example displays the contents of TEXT by specifying a path starting with the parent directory (..):

dir ../text

directory of ../text 14:06:47
manual

The following command changes the current data directory from PROG to ELLEN:

chd .../ellen

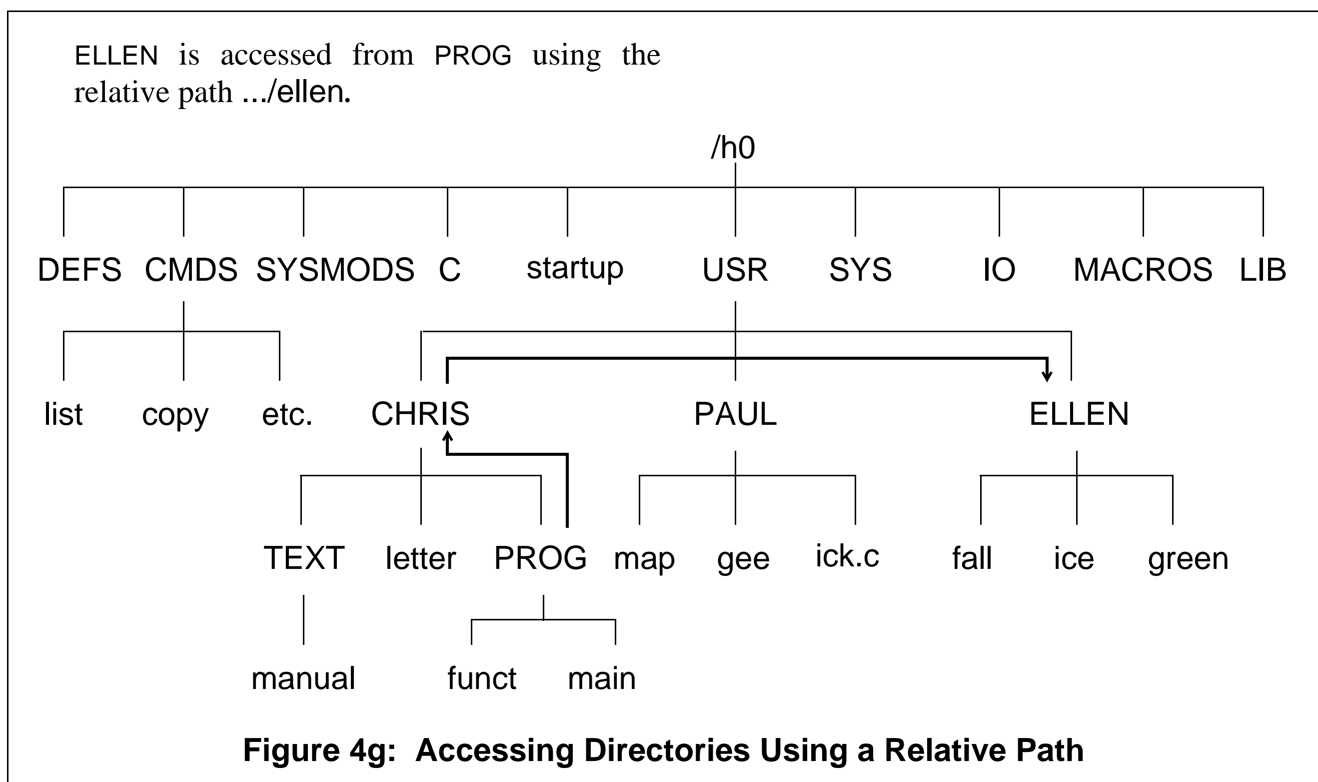


Figure 4g: Accessing Directories Using a Relative Path

You can use any number of periods (.) to access higher directories. One period is added for each additional level. An error is not returned if you specify a greater number of directory levels above your current data directory than actually exist. Instead, this indicates the root directory on your system. For example, this command displays the contents of the root directory:

dir ....

This may be helpful if you are not sure how far down you are in the directory structure. The next example changes your current data directory from PROG to MACROS:

chd ..../macros

Using the Pd Utility

When the file system becomes complex, you may become confused as to where the directory you are currently working in is located in relation to the overall file system.

> + The pd utility displays the complete pathlist from the root directory to your current data directory.

For example, if your current data directory is PAUL:

pd
/h0/USR/PAUL

Likewise, if you forget which directory is your current execution directory, type pd -x to display the pathlist to the current execution directory.

Using Makdir to Create New Directories

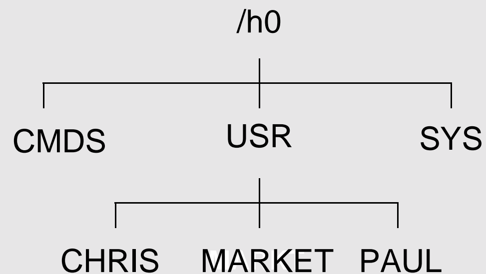
You create new directories using the makdir utility. For example, to create a directory called BUS.DEPT, type:

makdir BUS.DEPT

BUS.DEPT now is a new entry in your current directory.

If you want the new directory created somewhere other than your current directory, you must specify a pathlist. For example, makdir /h0/usr/BUS.DEPT creates the new directory in USR.

> + The makdir /h0/usr/MARKET command creates a new directory called MARKET in the USR directory.



Rules for Constructing File Names

When creating files and directories, you must follow certain rules. Any file name can contain from 1 to 28 upper or lower case letters, numbers, or special characters as listed below. While the file name may begin with any of the following characters or digits, each file name must contain at least one letter or number. Within these limitations, a name can contain any combination of the following:

upper case letter:	A - Z	underscore:	_
lower case letter:	a - z	period:	.
decimal digits:	0 - 9	dollar sign:	$

File names may not contain spaces. Instead, use the underscore (_) or the period (.) to improve the readability of file and directory names. OS-9 does not distinguish upper case letters from lower case letters. The names FRED and fred are considered the same name.

> + **NOTE:** By OS-9 convention, directory names are in upper case and file names are in lower case. This allows you to easily distinguish directories from files. This is only a recommendation for easy use; you may develop your own style.

Here are some examples of legal names:

raw.data.2	**project_review_backup**
X6809	**$SHIP.DIR**
...c	**12345**

Here are some examples of illegal names:

Max*min	** is not a legal character*
open orders	*name cannot contain a space*
this.name.obviously.has.more.than.28.characters	*too long*

NOTE: File names that start with a period are not displayed by dir unless the -a option is used. This allows you to hide files within a directory.

Creating Files

You can create files in many ways. Text files are generally created with the build utility, the edt utility, or the µMACS text editor. These file building tools are provided with the Professional OS-9 package for your convenience.

Use the build utility to create short text files. To use the build utility, type build, followed by the name of the file you want to create. build responds with the prompt:

?

This tells you that build is waiting for input. To terminate build, type a carriage return at the ? prompt. For example:

```
$ build test
? Some programmers have been known to
? howl at full moons.
?
$
```

You cannot edit files with build.

You may also use the edt utility to create files. edt is a line-oriented text editor that allows you to create and edit source files. To use the edt utility, type edt and the desired pathlist. If the file is new or cannot be found, edt creates and opens the file. edt then displays a question mark (?) prompt and waits for an edit command. If the file is found, edt opens it, displays the last line, and then displays the ? prompt. edt is fully detailed in the ***OS-9 Utilities*** section.

The preferred method of creating and editing files is with µMACS. µMACS is a screen-oriented text editor designed for creating and modifying test files and programs. Through the use of multiple buffers, µMACS allows you to display different files or different portions of the same file on the same screen. In addition, extensive formatting commands allow you to reformat paragraphs with new user-defined margins, transpose characters, capitalize words, and change words or sections into upper or lower case. For a more detailed description, see the ***Using µMACS*** manual.

Examining File Attributes with Attr

When you create a file using build or µMACS, only the owner read and owner write permissions are set. When you create a directory, it initially has all the permissions set except the single user permission.

To examine file attributes, use the attr utility. To use this utility, type attr, followed by the name of a file. For example:

```
$ attr newtest
------wr
```

The file newtest has the permissions set for owner reading and owner writing. Access to this file by anyone other than the owner is denied.

> + **Just a reminder:** Users with the same group.user ID as the person who created the file are considered owners. However, if the file is created by a group 0 user, only users in the super group can read, write, or execute the file.

If you use attr with a list of one or more attribute abbreviations, the file's attributes are changed accordingly, provided you have the proper write permission to access the file. The attribute abbreviations do not have to be listed in any particular order. The letter n preceding an attribute removes that permission.

The following command enables public read and write permission and removes execution permission for both the owner and the public:

$ attr newtest -pw -pr -ne -npe

If you are the owner of a file, you can change the access permissions regardless of what the permissions indicate. Thus, the owner always has the right to delete a file, change the user privileges, etc. Users in the same group have the same permissions as the owner.

The directory attribute is somewhat different than the other attributes. It could be dangerous to be able to change directory files to normal files or a normal file to a directory. For this reason, you cannot use attr to turn the directory (d) attribute on; use makdir to turn this attribute on. Furthermore, you can only use attr to turn the directory attribute off if the directory is empty.

Listing Files

Use the list utility to display the contents of files. By default, list displays the lines of text on your terminal screen. To examine a file, type list, followed by the name of the file. For example:

```
$ list test
Some programmers have been known to
howl at full moons.
$
```

It is important to remember that you cannot list a directory. If you type the command list USR, the following error message and error number are returned:

list: can't open "USR". Error# 000:214.

This means that you cannot access USR because it is a directory.

list displays text files. All distributed files in CMDS are executable program module files. If you try to list the contents of a random access data file or an executable program module file, you see what appears to be random data displayed on your screen. This may also include unprintable characters, such as escape or delete, that could change your terminal's operating parameters. If the operating characteristics of your terminal are affected, first try turning the terminal off and on. If this does not re-initialize the terminal, consult your terminal operating manual.

Copying Files

Use the copy utility to make a duplicate of a file. To copy a file, type copy, followed by the name of the file to be copied, followed by the name of the duplicate file. For example:

$ copy test newtest

If you list the file newtest, it is an exact copy of test.

The file you are copying and the duplicate file may be located in any directory; they do not have to be in your current data directory. For files located outside of your current data directory, you may use full or relative pathlists. The following example uses Figure 4h. The first command copies the file gee in the PAUL directory to a file named new.info in the TEXT directory:

copy /h0/usr/paul/gee /h0/usr/chris/text/new.info

Assuming your data directory is USR, the following commands would have the same effect:

copy /h0/usr/paul/gee chris/text/new.info
copy paul/gee chris/text/new.info

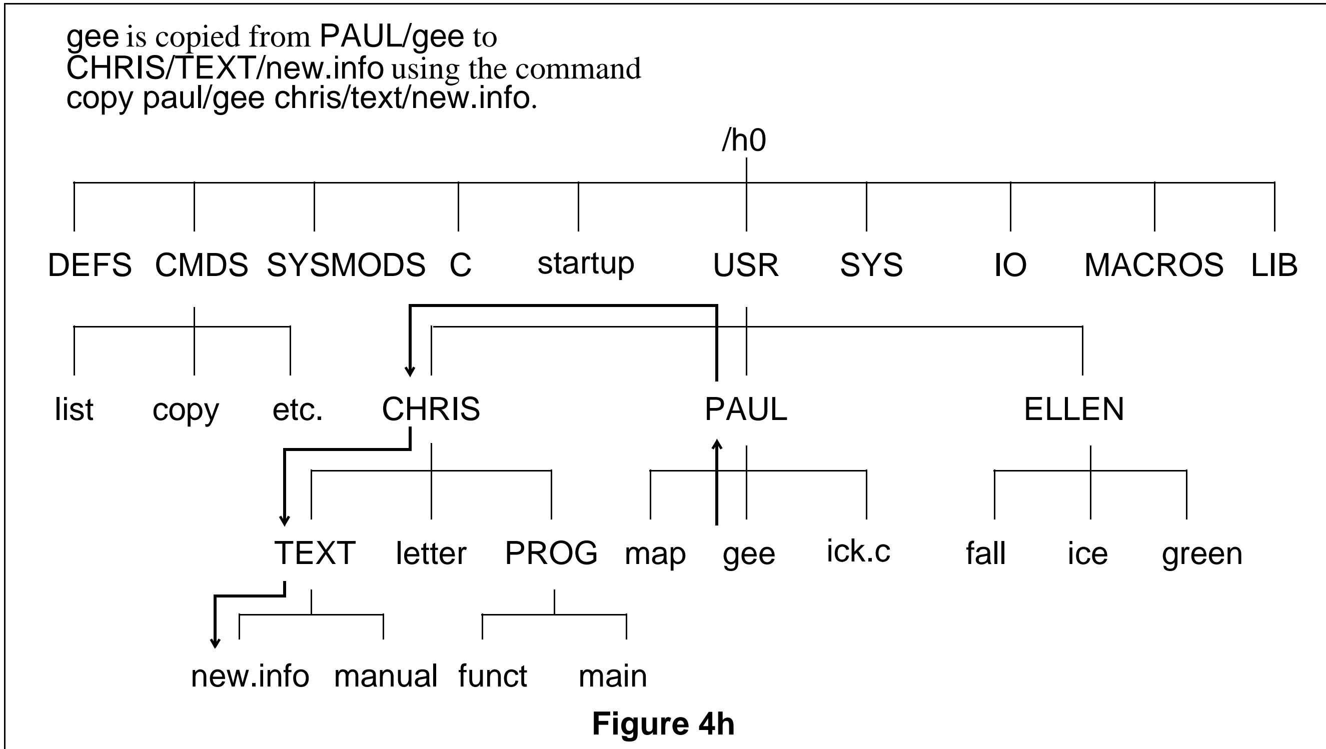


Figure 4h

If you try to copy the contents of one file into an existing file, you will receive Error #000:218 Tried to create a file that already exists. If you know the file exists but you want to overwrite it anyway, use the -r option. For example, the following command replaces the contents of green with the contents of fall.

```
$ copy fall green -r
```

If you list the contents of both files, you will see that they are identical.

At some point, you may want to copy more than one file at a time into another directory. By using the -w=<dir> option of copy, you can copy more than one file with a single command. For example, if your current directory is PROG and you want to copy all of the files in PROG into the TEXT directory, you could type the following command line:

```
$ copy * -w=../text
```

This option will print the name of the file after each successful copy. If an error occurs, the prompt continue (y/n) is displayed.

Remember that an asterisk is a wildcard. For more information about wildcards, refer to the section on wildcards in the chapter on the shell.

> + copy uses a 4K memory buffer by default. This means that only 4K of information is read from the original file and written to the new file at one time.

If you have a large file, the copy procedure may be slow because the system has to perform multiple read and write statements. The -b option may be used to increase the buffer size. This would make the copy procedure faster for large files. To use the -b option, type copy, the original file name, the new file name, and -b=<num>k.

For example, typing copy gee mine -b=20k allocates a 20K buffer for copying the file gee into the file mine.

NOTE: You must have permission to copy the file. That is, you must be the owner of the file to be copied or the public read permission must be set in order to copy the file. You must also have permission to write in the directory you specify. In either case, if the copy procedure is successful, the new file has your group.user number unless you are the super user. If you are the super user, the new file will have the same group.user number as the original file.

For more information concerning copy, refer to the ***OS-9 Utilities*** section.

Dsave: Copying Files Using Procedure Files

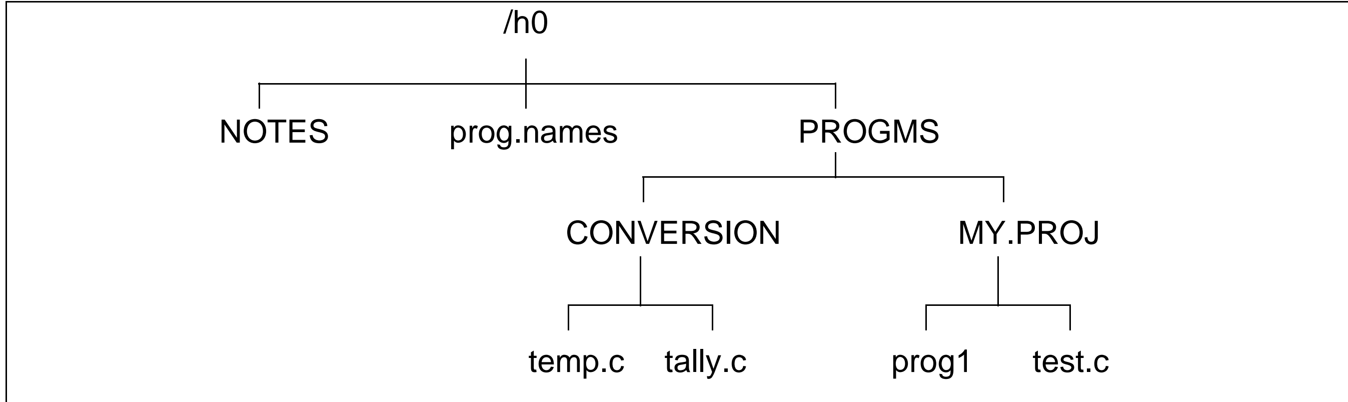
Use the dsave utility to copy all files and directories within a specified directory by generating a procedure file. The procedure file is either executed later to actually perform the copy or, by specifying the -e option, executed immediately.

NOTE: A procedure file is a special OS-9 file. It contains OS-9 commands. Each command is specified on a line, one command per line. When the procedure file is executed, the OS-9 commands it contains are executed in the order they are listed in the procedure file. Procedure files are discussed in more detail in the chapter on the shell.

+ To use the dsave utility, type dsave followed by the pathlist of the directory into which the files are copied, followed by any options you wish to use.

If no pathlist is specified for the destination, the files are copied to the current data directory at the time the procedure file is executed. If you do not specify the -e option or redirect the output to a file, dsave sends the output to the terminal.

The example below uses the following directory structure:



If PROGMS is your current data directory and you type dsave ../notes, the following appears on your screen:

```
$ dsave ../notes
-t
chd ../notes
tmode -w=1 nopause
load copy
Makdir MY.PROJ
Chd MY.PROJ
Copy -b=10 /h0/PROGMS/MY.PROJ/prog1
Copy -b=10 /h0/PROGMS/MY.PROJ/test.c
Chd ..
Makdir CONVERSION
Chd CONVERSION
Copy -b=10 /h0/PROGMS/CONVERSION/temp.c
Copy -b=10 /h0/PROGMS/CONVERSION/tally.c
Chd ..
unlink copy
tmode -w=1 pause
$
```

Because the output was not redirected to a procedure file and the -e option was not used, the above commands were not executed. They were just echoed to your screen.

If you now type dsave ../notes -e, the commands are again echoed to the screen. However, the contents of the PROGMS directory are copied into the NOTES directory.

You can also redirect the output of dsave to a file. When you redirect the output, the commands that are output from dsave are essentially captured in a file. You can later execute this file to actually perform the dsave operation.

To redirect the output from dsave to a file, use the redirection modifier for standard output. The standard output modifier is the > symbol.

For example, from the PROGMS directory, you can redirect the output from dsave into a file called make.bckp by typing:

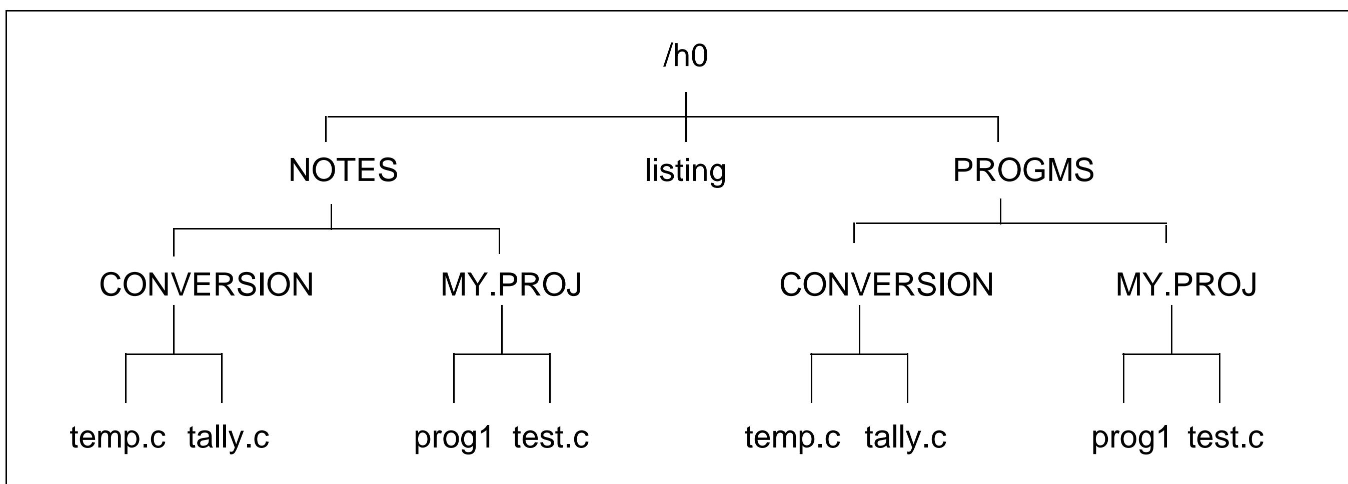
dsave >make.bckp

This command creates make.bckp in the current data directory. To perform the dsave, type make.bckp at the command line.

Redirecting the output to a file is helpful when you want to save most, but not all, of the files in the directory or directories being saved. You can edit make.bckp before performing the dsave. This allows you to save only selected files.

Regardless of how you decide to perform the dsave, if dsave encounters a directory file, it automatically creates a new directory and changes to that directory before generating copy commands for files in the subdirectory.

In the dsave example, the directory structure looks like the following after dsave has finished:



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

If an error occurs during the dsave process, the following prompt is displayed:

continue (y,n,a,q)?

A y indicates that you wish to continue with dsave. An n indicates that you do not wish to continue with dsave. An a indicates that all possible files should be copied and the prompt should not be displayed on error. A q indicates that you want to exit the dsave procedure.

If for any reason you do not wish to be bothered by the prompt, the -s option is available. This skips any file which cannot be copied and continues the dsave routine without the error prompt.

When you copy several subdirectories, you can use the -i option to indent for directory levels. This helps to keep track of which files are located in which directories.

You can use dsave to keep current directory backups. Use the -d or -d=<date> options to compare the date of the file to be copied with a file of the same name in the directory where it is to be copied. The -d option copies any file with a more recent date. The -d=<date> option copies any file with a date more recent than that specified. The following example shows the use of dsave with the -d option:

```
$ chd /d0/BACKUP
$ dir
                              Directory of . 14:14:32
 Owner   Last Modified  Attributes Sector Bytecount Name
-------  -------------  ---------- ------ --------- ----
 12.4    90/11/12 1417  ------wr   20CO     11113 program.c
 12.4    90/10/05 1601  ------wr   313D      5744 prog.2
$ chd /d0/WORKFILES
$ dir
                              Directory of . 14:14:32
 Owner   Last Modified  Attributes Sector Bytecount Name
-------  -------------  ---------- ------ --------- ----
 12.4    90/11/12 1417  ------wr   DODO     11113 program.c
 12.4    90/11/12 1601  ------wr   3458      5780 prog.2
$ dsave -deb32 /d0/BACKUP
$ chd /d0/BACKUP
$ dir
                              Directory of . 14:14:32
 Owner   Last Modified  Attributes Sector Bytecount Name
-------  -------------  ---------- ------ --------- ----
 12.4    90/11/12 1417  ------wr   5990     11113 program.c
 12.4    90/11/12 1601  ------wr   A12B      5780 prog.2
```

Only prog.2 was copied to the BACKUP directory because the date was more recent in the WORKFILES directory.

For more information about dsave, refer to the ***OS-9 Utilities*** section.

Del and Deldir: Deleting Files and Directories

Use the del and deldir utilities to eliminate unwanted files and directories. If you no longer need a file, deleting the file frees disk space. You *must* have permission to write to the file or directory in order to delete it.

- del deletes a file.
- deldir deletes a directory.

To delete a file, type del, followed by the name of the file that you want deleted. For example, to delete the file test that you created with build, you would type:

del test

If you execute dir you see that test is no longer displayed.

When deleting files, you may use wildcards. For example, if you have three files, trial, trial1, and trial.c, in a directory and you want to use wildcards to delete trial and trial1, you may be tempted to type del trial*, but this would also delete trial.c, a file you want to keep. **Use caution when you use wildcards with utilities like del and deldir**. It is easy to unintentionally delete files you want to save.

NOTE: Wildcards are discussed in the chapter on the shell.

The del -p option displays the following prompt before deleting a file:

delete <filename> ? (y,n,a,q)

Type y to delete the file; n if you do not want to delete the file; a if you want to delete all specified files without further prompts; and q if you want to quit the deleting process. This helps prevent deleting files you want to keep.

Deleting a directory is a little different. Use the deldir utility to delete directories. deldir first deletes all the files and directories in the given directory, and then, if no errors occur, finally deletes the directory name. For example:

$ deldir USER2

Deleting directory: USER2
Delete, List, or Quit (d, l, or q) ?

At the prompt, type l to list the contents of the directory, d to delete the directory, or q to quit and not delete anything.

Just a reminder: Never delete a file or directory unless you are sure you do not need it.

End of Chapter 4