T H R E E

Working with Files and Directories

▲Chapter Syllabus

For a user of a UNIX system, dealing with files is a routine task. A considerable portion of time is consumed dealing with files. To make productive use of your time, you always need to reduce time spent on file handling. This can be accomplished by efficient use of commands related to file and directory operations. HP-UX provides a simple and powerful command mechanism. Other than normal use of file handling commands, you can group these commands and pass output of one command to another. Operations can be performed on single files or on a group of files at the same time. When preparing for the HP-UX certification examination, these basic file operations are important. There are some questions on the certification exam directly related to file handling. Other than that, many other questions have implicit application to the commands presented here.

Most of this chapter is devoted to use and explanation of file handling commands. You will learn

very basic operations on files. These operations include creating, listing, deleting, and displaying contents of a file. Then you will learn some rules that apply to filenames, and you will see which filenames are legal in HP-UX. After that, basic operations on directories is presented, where you will find commands for creating, deleting, and changing directories. Copying and moving files are routine tasks and commands for these are presented next. For operations on multiple files simultaneously, wildcards are used, and you shall see the use and purpose of each of these wildcards. Searching text files for a particular string and finding files with particular names on disk will be the next topic. At the end of the chapter, you learn more commands for file manipulation.

3.1 Basic Operations on Files

The most common operations on files are creating new files, deleting unnecessary files, listing filenames, and displaying contents of a file. After login, you can perform all of these operations in your home directory. Let us see how we do it.

Creating a File

A file is a named area on the disk(s) where you can store some information. You can create files in many different ways. Some of the common methods of creating files are:

1. Using I/O redirection with the help of a command such as `cat`.
2. Using the `touch` command.
3. Using an editor.
4. Using programs that create files during their operation.

In this section, you learn how to use the `cat` and `touch` commands to create new files.

USING THE `cat` COMMAND TO CREATE FILES

The `cat` command is a basic command used for creating new files containing text. For example, if you want to create a file that contains three lines and is named `newfile`, the process is as follows. First of all, use the `cat` com-

mand with the "greater than" sign (>) and the filename as the command line argument. Then, type the three lines of text. Last, press Ctrl+D.

```
$ cat > newfile <ENTER>
This is the first line. <ENTER>
This is the second line. <ENTER>
This is the third and last line. <ENTER>
CTRL-d
$
```

Note that you press Enter at the end of each line. When you have finished entering the text, you press Ctrl+D (this means press the "Control" and "D" keys simultaneously) to end the text entry process and save the file. Note also that this key sequence sends an EOF (End Of File) character.

The method used here for creating the file is called I/O redirection. You can use many other commands to create files. I/O redirection is discussed in more detail in Chapter 5.

Please note that use of the `cat` command for creating a new file is not very common but it is the simplest way to do so. Most of the time you will be using the `vi` editor to create or modify files. The `vi` editor is discussed in more detail in Chapter 6.

USING THE `touch` COMMAND TO CREATE FILES

The `touch` command creates an empty file if one doesn't already exist. An empty file is one that exists on disk but has no data inside it. The following command creates a file named `myfile` with zero size:

```
touch myfile
```

The `touch` command is useful when you are dealing with programs that need some files to be present for their operation. Using `touch`, you can create these files.

Note that if you use the `touch` command on an existing file, `touch` updates the modification time for this file to the current time. In summary, if you use the `touch` command with an argument that represents an existing file, the command updates only the modification time for the file without making any change to its contents. If you use the `touch` command with an argument that shows a nonexistent file, the command creates the file with zero byte size.

Listing Files

Now that you have created a file, you can verify the file by listing it using the `ls` command:

```
$ ls newfile
newfile
$
```

The `ls` command showed that our newly created file `newfile`, does exist and that the file creation process was successful. What if you want to see a list of all other files? This is simple; you use the `ls` command without any argument.

```
$ ls
file1       file2       myfile.zip  newfile
$
```

Now the `ls` command shows there are four files with the names shown above. Please note that UNIX is case sensitive, meaning it differentiates between lowercase letters and uppercase letters, so the filename `myfile.zip` is different from `MyFile.zip`.

HP-UX has another popular command to list files. This is the `ll` (Long Listing) command.

```
$ ll
total 0
-rw-rw-rw-   1 rr         users           23 Jun 17 20:57 file1
-rw-rw-rw-   1 rr         users          531 Jun 17 20:57 file2
-rw-rw-rw-   1 rr         users            3 Jun 17 20:58 myfile.zip
-rw-rw-rw-   1 rr         users            9 Jun 15 20:01 newfile
$
```

This command shows there are four files, the names of which are displayed in the last column of the output. If you are wondering what the "`-rw-rw-rw-`" characters displayed in the first column are, these are the file permissions showing who is allowed to read, write, and execute each file. We discuss file permissions in more detail in Chapter 8. The `ll` command is another example of how some commands are linked to other commands, which was discussed in Chapter 2. This command is linked to the `ls -l` command. The `ls` command has many options, `-l` being one of them, and you can have a look at these using its manual pages.

Now we determine the other columns in the file listing. The second column shows how many links are associated with this file. A "1" (numeric one)

means there is no other link to this file. The next column shows the owner of the file. The word "users" is the group name of the user "rr," who owns this file. The next column shows the file size in number of bytes. Then we have the date and time of the last modification to the file. And in the last column, the name of the file is displayed.

Deleting Files

To keep the system clean, you need to delete unwanted files from time to time. The files are deleted with the `rm` command.

```
$ rm newfile
$
```

The `rm` command normally provides no output. You need to be careful when deleting files, as the deleted files cannot be undeleted. An error message is displayed only if the file you are deleting does not exist or you don't have permission to delete a file.

Displaying Contents of a Text File

There are many ways to display the contents of a text file. You can use different commands as well as an editor to display files. In this section, you learn some basic commands that can be used to display text files.

USING THE `cat` COMMAND TO DISPLAY A TEXT FILE

We have already used the `cat` command for creating new files. The same command is used to display contents of text files.

```
$ cat newfile
This is the first line.
This is the second line.
This is the third and last line.
$
```

We have just omitted the ">" symbol from the command line that we used to create the file. The `cat` command displays the entire contents of a file in one step, no matter how long the file is. As a result, the user is able to see only the last page of displayed text.

USING THE `more` COMMAND TO DISPLAY TEXT FILES

There is another useful command called `more` that displays one page of text at a time. After displaying the first page, it stops until the user hits the Space Bar. The `more` command then displays the next page of text, and so on. Figure 3.1 shows a screenshot of the `more` command while displaying the `/etc/profile` file.

You can also use the Enter key to display only the next line instead of the whole next page. To quit the `more` command, press the Q key at any time.

Note that you can also use the more command to search a text string in the file being displayed. For example, if you want to search the file for the text string "mail," type the slash character followed by the word you want to search. When you type this word, it will be displayed on the bottom line. After you are finished typing, press the Enter key and the text scrolls to the line where the word "mail" is present. The text scrolls automatically so that this line appears second from the top. If you want to search a string consisting of multiple words, just type the whole string after the slash character. To search the same word again, you don't need to type it again; just press the slash key followed by the Enter key and the previous search is repeated.

There are many commands used during the display of a file. These commands are used after starting the `more` command. The most commonly used commands consist of a single keystroke. Table 3.1 shows widely used commands and their descriptions.

```
# @(#)B.11.11_LR

# Default (example of) system-wide profile file (/usr/bin/sh initialization).
# This should be kept to the bare minimum every user needs.

# Ignore HUP, INT, QUIT now.

        trap "" 1 2 3

# Set the default paths - Do NOT modify these.
# Modify the variables through /etc/PATH and /etc/MANPATH

        PATH=/usr/bin:/usr/ccs/bin:/usr/contrib/bin
        MANPATH=/usr/share/man:/usr/contrib/man:/usr/local/man

# Insure PATH contains either /usr/bin or /sbin (if /usr/bin is not available).

        if [ ! -d /usr/sbin ]
        then
                PATH=$PATH:/sbin

        else    if [ -r /etc/PATH ]
                then
profile (23%)
```

Figure 3.1 *Use of the* `more` *command.*

Table 3.1 *Commands Used within the* `more` *Command Session*

Command	Description
f	Forward one page of text (moving down).
b	Backward one page of text (moving up).
d	Down half page of text.
u	Up half page of text.
g	Go to first page of text.
G	Go to last page of text.
r	Repaint screen.
h	Display a help screen with list of all commands.
:f	Display statistics of current file (filename, lines, characters).
q	Quit the more command session.

As you can see from Table 3.1, there are many things you can do with the `more` command.

USING THE `pg` COMMAND TO DISPLAY TEXT FILE

The `pg` (page) command is similar to the `more` command with a few differences. The command is used for page display and has some useful features. A typical screen for the command `pg /etc/profile` is shown in Figure 3.2.

```
# @(#)B.11.11_LR

# Default (example of) system-wide profile file (/usr/bin/sh initialization).
# This should be kept to the bare minimum every user needs.

# Ignore HUP, INT, QUIT now.

        trap "" 1 2 3

# Set the default paths - Do NOT modify these.
# Modify the variables through /etc/PATH and /etc/MANPATH

        PATH=/usr/bin:/usr/ccs/bin:/usr/contrib/bin
        MANPATH=/usr/share/man:/usr/contrib/man:/usr/local/man

# Insure PATH contains either /usr/bin or /sbin (if /usr/bin is not available).

        if [ ! -d /usr/sbin ]
        then
                PATH=$PATH:/sbin

        else    if [ -r /etc/PATH ]
                then
:
```

Figure 3.2 *A typical screenshot of the* `pg` *command.*

As with the `more` command, you can use the `pg` command for:

- Displaying a full page of text at one time.
- Searching through text.
- Scrolling back and forth through text.

A colon character is displayed at the end of each screen of text. You can use the Enter key to move to the next page. When you are on the last page of text, you will see ":EOF" string that shows the End Of File (EOF). Pressing the Enter key at this point will terminate the command.

You can move back and forth in the `pg` command. Entering "`d`" and then pressing the Enter key will move one page forward. If you want to go back one page, enter `-d` and then press the Enter key. You can also scroll multiple pages by including a number with the `d` command. For example, if you type in "`3d`" and then press the Enter key at the colon prompt, you will move forward three pages. Entering "`-2d`" and then pressing the Enter key will move backward two pages (in other words, move two pages up).

USING THE `pr` COMMAND TO DISPLAY TEXT

You can also use the `pr` command to display contents of a file to standard output. The `pr` command separates text into pages and prints date, filename, and page number on top of each page. The following command displays contents of `/etc/profile` on your display screen.

```
pr /etc/profile
```

Note that when you learn I/O redirection later in this book, you can easily redirect output of this command to a printer.

3.2 File Naming Rules

When you create a new file, there are some rules that govern the naming of the file. These rules are related to the length of the filename and the characters allowed in the name of a file.

General Guidelines for Filenames

Generally, a filename in UNIX can be as long as 256 characters. The rules that apply to filenames are as follows:

1. A filename can be a combination of alphabetic, numeric, or special characters.

2. All alphabetic characters, both upper- (A … Z) and lowercase (a … z) can be used in filenames.
3. Numbers from 0 to 9 can be used in filenames.
4. Special characters such as the plus sign (+), minus sign (-), underscore (_), or dot (.) can be used in filenames.
5. As mentioned earlier, since UNIX is case sensitive, uppercase and lowercase letters are treated separately. So, filenames `myfile`, `Myfile`, `MyFile`, and `myfilE` are all different names.
6. There are no special names for executable files in UNIX. The file permissions show which files are executable and which are not.

It is recommended that you use only alphabetic/numeric characters and the underscore character in filenames. Although you can use other special characters in filenames, they have other meanings for the shell and may create confusion when you are performing operations on files.

Hidden Files

Any filename that starts with a dot (.) is not displayed when using the `ll` or `ls` commands. These are hidden or invisible files. Usually, these files are used to store some kind of configuration information and reside in the home directory for a user. The user startup file with the name `.profile`, if you remember, is a hidden file. To display the hidden files, use the `ls -a` command.

```
$ ls -a
.profile    newfile     testfile.zip
$
```

Hidden files are safe from the `rm` command when it is used to delete all files in a directory. The `rm` command does not delete hidden files unless explicitly directed to do so, such as with the `rm .profile` command, which will delete the `.profile` file.

Note that while using commands such as `ll` or `ls`, the Superuser will see hidden files by default .

3.3 Working with Directories

Directories are placeholders for files. As with files, basic operations on directories are creating new directories, deleting directories, and moving from one directory to another in a directory hierarchy. Commands used for these

operations are presented in this section. As far as names of directories are concerned, rules that apply to ordinary files are also applicable here.

Creating Directories

A directory can be considered a special type of file that is used as a folder to keep other files and directories inside it. Directories are used to organize files in a logical and manageable way. A directory can be created with the `mkdir` command. The following command creates a directory with the name `newdir`.

```
$ mkdir newdir
$
```

After creating a directory, verify its existence with the `ls` or `ll` command. Note that when we use the `ll` command for long listing, the first character in the file permissions is "`d`" instead of "`-`", showing that it is a directory, not an ordinary file.

```
$ ll
total 8
-rw-rw-rw-   1 rr         users            20 Jun 18 10:23 file1
-rw-rw-rw-   1 rr         users            99 Jun 18 10:23 file2
-rw-rw-rw-   1 rr         users            21 Jun 18 10:24 myfile.zip
drwxrwxrwx   2 rr         users            96 Jun 18 10:23 newdir
-rw-rw-rw-   1 rr         users           552 Jun 18 10:24 newfile
$
```

Using the `ls` command without the `-l` option shows all names of files and directories and does not distinguish between them. If you don't want to display the long listing and still need to distinguish between files and directories, you can use either the `lsf` or the `ls -F` command. These are equivalent commands and the screen output simply appends a "`/`" symbol at the end of the directory name.

```
$ lsf
file1        file2        myfile.zip   newdir/      newfile
$
```

Here you can see that `newdir` is a directory, while the other four files are ordinary files. You may see an asterisk at the end of some filenames. This sign shows that the file is an ordinary file and is executable.

If you want to list information about the directory itself, you can use the `-d` command line option. The following command lists the `/etc` direc-

tory. Note that if you omit this option, all files inside the `/etc` directory will be listed.

```
$ ll -d /etc
dr-xr-xr-x  31 bin        bin           6144 Nov 16 10:03 /etc
$
```

Deleting Directories

Directories are deleted with `rmdir` command. By default, the command deletes only empty directories. If the directory contains other files or directories, you can use `rm -rf <dirname>` to delete it. If you need to delete a directory that is not empty and you want to interactively select which files to be deleted, use the `rm -ri` command that runs the `rm` command interactively and it will ask you which files to delete. Now if you select any file not to be deleted, the directory will not be deleted because it is not empty. The following sessions show that there is one file inside the `newdir` directory. The name of this file is `file1`. When prompted, you selected not to delete this file. The result is that the directory `newdir` is not deleted because a file exists inside this directory.

```
$ rm -ri newdir/
directory newdir: ? (y/n) y
newdir/file1: ? (y/n) n
newdir: ? (y/n) y
rm: directory newdir not removed.  Directory not empty
$
```

Note that the characters in boldface in the above listing are those that you type during the execution of the command. Select "`y`" for yes and "`n`" for no.

Also note that if you don't use the `-i` command line switch, the directory `newdir`, including all other files inside it, will be deleted without a prompt.

You can also use the `-f` command line switch to override directory permissions (if you are the `root` user) to delete files and directories.

Be careful when using `rm -rf`, as it removes the entire directory tree without any warning to the user.

Understanding Directory Structure

A UNIX file system is composed of directories and files. The top-level directory is called the *root directory* and is represented by a forward slash " / " symbol. All other directories and files may be considered *inside* the root directory. The *current directory* is what ever directory you are in presently. The current directory path can be displayed using the `pwd` command at any time. A directory that is one level above the current directory is called a *parent directory* while a directory one level below the current directory is called a *child directory*. In Figure 3.3, the root directory (/) is the parent directory for `home` directory and `boota` is a child directory of `home` directory.

Parent directories and child directories are relative to each other. For example, the `home` directory is a child directory of the `root` directory, but it is a parent directory for the directory `boota`.

The directory names are referenced relative to the `root` directory. A complete reference name to a directory is called a *path name*. For example, the path name of the `home` directory is `/home`. Similarly, the path name of the directory `boota` is `/home/boota`. It is easy to judge from the path name that `boota` is a child directory of `home`, which in turn is a child directory of the `root` directory. Files also have path names similar to directories. For example, a complete path name for a file created in the directory `/home/boota` with the name `myfile` is `/home/boota/myfile`. A path name that starts with the "/" symbol is called the *absolute path* name.

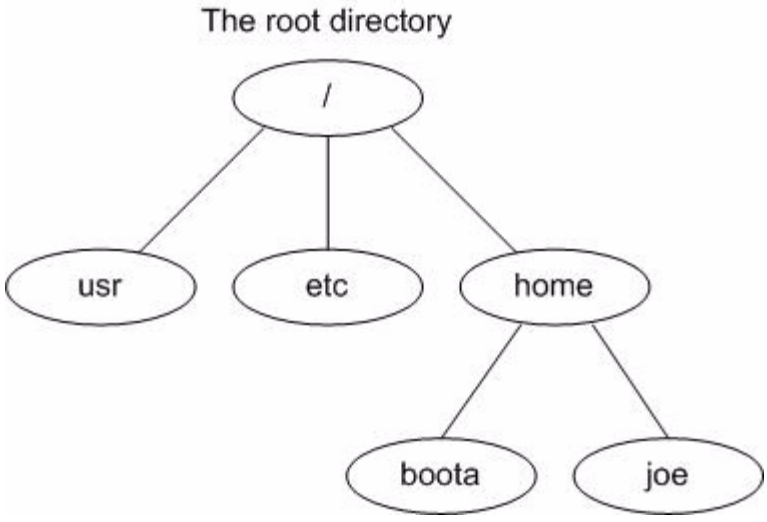

Figure 3.3 *A sample directory tree.*

You can also use *relative path* names, which start from the current directory. For example, to refer to a file with the name `alpha` in the parent directory of the current directory, we may use the path name `../alpha`.

When ever a new directory is created, two entries are automatically created in the new directory. These are "." and ".." where "." is a reference to the current directory and ".." is a reference to the parent directory of the current directory.

You can cascade the double dot character to as many levels as you want. For example, if you are in the `/home/boota` directory, reference to the file `myfile.txt` in the root directory can be made by specifying the path `../../myfile.txt`. Note that the first pair of dots on the left side of filename refers to the parent directory (in this case, `/home`) while the second pair refers to the parent of the parent directory (in this case, "/"). As another example, if you want to copy the file `/etc/profile` to the current directory, `/home/boota`, all of the following commands are valid.

```
cp /etc/profile .
cp ../../etc/profile .
cp ./../../home/../etc/profile /home/boota
```

The first command uses an absolute path for the source file and uses a single dot character to indicate the destination file location, which is the current directory.
The second and third commands use double dot characters to refer to different directories in the directory structure. Ultimately, all commands refer to the same source file. The `cp` command is discussed in detail shortly.
As another example, all of the following commands have the same effect and they all display the contents of the `/etc/profile` file, assuming you are in `/home/boota` directory.

```
cat /etc/profile
cat ../../etc/profile
cat ./../../home/../etc/profile
```

It is interesting to note that system administrators usually refer to the current and parent directories as "dot" and "dot-dot," respectively. Sometimes the parent directory is referred to as "double-dot."

Moving Around in a Directory Tree

You used the `pwd` command in Chapter 2. This command was used to display the path to the current directory. The `cd` (Change Directory) command

is used to go to some other directory in the directory tree. This command, like other UNIX commands, can be used with both absolute and relative path names. You already know that a user automatically goes to the home directory just after the login process. We again consider the example of the user `boota`, who has just logged in and is in his home directory, `/home/boota`. To first confirm that he is indeed in his home directory and then move to the `/etc` directory, the user issues the following commands:

```
$ pwd
/home/boota
$ cd /etc
$
$ pwd
/etc
$
```

The last `pwd` command shows that the user has moved to the destination directory, `/etc`. In this example, we used an absolute path with the `cd` command. For an example of using relative path, consider that the user `boota` is in his home directory `/home/boota` and wants to move to `/home`, the parent) directory. Now he can use either the `cd ..` or `cd /home` commands; both will have the same effect. In `cd ..`, he asked the shell to move to the parent directory of the current directory. What if you use `cd ../..`?

3.4 Copying and Moving Files

Many times you will be copying or moving files from one place to another. These two operations are similar except that the old file is deleted in the move operation. An important thing to remember about copy and move operations is that the move process retains ownership and permission of the destination file. The copy operation, by default, makes the current user the owner of the destination file.

Copying Files

The files are copied with the `cp` command. The source and destination filenames are supplied to the `cp` command as arguments. The first argument is the source filename and the second argument is the destination filename. The following is an example of the `cp` command:

```
$ cp myfile anotherfile
$
```

This command copies `myfile` from the current directory to `anotherfile` in the current directory. Note that if the destination filename already exists, it will be overwritten without a warning.

It is possible to copy files from any directory to any other directory (with write access) using the path names of the files. For example, if you want to copy the file `profile` from the `/etc` directory (`/etc/profile`) to the current directory with the name `myprofile`, the command line will be as follows:

```
$ cp /etc/profile myprofile
$
```

If you want to copy the file in the above example with the same name in the current directory, you simply use "`.`" in place of the destination name. Note that the "`.`" character is a relative path that refers to the current directory. For example, to copy the `/etc/profile` file with name `profile` in the current directory, the following command can be used:

```
$ cp /etc/profile .
$
```

Two or more files can be copied simultaneously using the `cp` command. In this case, the destination must be a directory name. The following command copies two files, `file1` and `file2`, from the current directory to the `/tmp` directory.

```
$ cp file1 file2 /tmp
$
```

Note that if the destination is a directory name, the source file is copied inside that directory. If the destination is a filename, the source file is copied as the destination file.

Moving and Renaming Files

The `mv` command is used for renaming files and moving files from one place to another in the directory structure. Like the `cp` command, it takes source and destination filenames as arguments. The first argument is the source file and the second argument is the destination file. If both source and destination names are specified without any path (absolute or relative), the file is

renamed. On the other hand, if one or both of the filenames contain a path that is not the same, the file is moved from the source location to the destination location.

RENAME A FILE

The following command renames the file "myfile" to "newfile."

```
$ mv myfile newfile
$
```

Make sure that the operation was successful by using the ll command.

MOVE A FILE

The following command moves a file named myfile from the current directory to the /tmp directory.

```
$ mv myfile /tmp/myfile
$
```

Note that the command "mv myfile /tmp" will do the same job as the previous command. However, if you want to move a file to a destination directory with a different name, the previous example is the right way to do it.

Two or more files can be moved simultaneously using the mv command. The destination must be a directory name in case you are moving multiple files. The following command moves two files, file1 and file2, to the directory /tmp.

```
$ mv file1 file2 /tmp
$
```

You must be careful with the mv command, because it will overwrite any existing file in the destination directory whose filename matches that of file being moved. And it will do so without warning. To ensure that no existing files are overwritten, always render the mv command as mv –i. In this case, if the destination file already exists, the mv command will ask you to confirm the move or rename operation.

Also note that the mv command is used to move and rename directories just as with files.

3.5 Wildcards

When you want to use many filenames in one command, such as in the case where `grep` (global regular expression print) is used to search a pattern in many files, it is inconvenient to type all of these filenames at the command line. *Wildcard* characters are used as a shortcut to refer to many files at one time. Two wildcards are used in UNIX: the asterisk character (`*`) and the question mark (`?`). The "`*`" matches zero or more different characters, whereas "`?`" matches only a single character. There is a third type of character-matching mechanism that checks ranges of characters. This is the `[]` pattern and the range of characters to be checked is specified inside the square brackets. Sometimes this is also called the third wildcard.

Use of "" as a Wildcard*

Suppose you use the `ls` command to list files and the following list appears:

```
$ ls
myfile   myfile00   myfile01   myfile010  myf      xyz
$
```

Now you can use the "`*`" character to list files you want to be displayed. If you want to list all files that start with the pattern "`myfile`," the command is as follows:

```
$ ls myfile*
myfile   myfile00   myfile01   myfile010
$
```

To list all files that start with "`my`," you use:

```
$ ls my*
myfile   myfile00   myfile01   myfile010  myf
$
```

Use of "?" as a Wildcard

The "**?**" matches only a single character. For example, if you want to list all files that start with "`myfile0`" and the last character may be any one, the result is as follows:

```
$ ls myfile0?
myfile00   myfile01
$
```

Now, try to figure out why "myfile010" did not appear in the list. This is because the "?" matches only one character. To list "myfile010," you need to match the last two characters.

The wildcard characters can be used wherever you need to specify more than one file. For example, if you want to copy all files (excluding hidden files) from the current directory to the /tmp directory, the command will be as follows:

```
$ cp * /tmp
$
```

Similarly, if you want to search for the word "home" in all files of the /etc directory, you can use the grep command as shown below. The grep command is used to search text and is covered in detail later in this chapter.

```
$ grep home /etc/*
```

The wildcard characters are very useful and if you master these, you can save a lot of time in your daily computer use.

Use of the "[ ]" as a Wildcard

This wildcard matches the range of characters specified inside the square brackets. Only one character from the range is taken. For example, [a-m] means any character between "a" and "m". Similarly, [acx] means any character from "a", "c" or "x". The following command lists all files in the /etc directory that start with either the "w" or the "x" character:

```
$ ls /etc/[wx]*
/etc/wall   /etc/whodo  /etc/wtmp   /etc/xtab
$
```

Note that you have used two wildcards in this command.

3.6 Special Directories

You have seen how to use wildcard characters to reference files and directories. There are some other methods to reference a few special directories. The following three rules apply when you reference special directories:

1. A dot (.)character is used to reference the current directory.

2. A double dot (..) character is used to reference the parent directory.
3. A tilde (~) character is used to reference the home directory of a user. This does not work if you are using Bourne shell. Since most of the HP-UX users use POSIX shell, the tilde character is used quite often.

For example, the following command copies the file `/etc/hosts` to the home directory of the current user:

```
cp /etc/hosts ~
```

The following command copies the same file to the current directory:

```
cp /etc/hosts .
```

If you are in the `/tmp/boota` directory, the following command will copy the file `/etc/hosts` to the `/tmp` directory (parent directory of the current directory):

```
cp /etc/hosts ..
```

3.7 Files Types

You have been using commands such as `cat` and `more` with text files. How do you know which file is a text file and which one contains binary data, and which one is a directory, or which one is a C program? There are multiple ways to determine a file type. Common file types used in HP-UX are listed below:

- Regular files are text files, binary files containing binary data (such as database files), and binary files that are executable.
- Directories are special files that contain other files and directories.
- Device files are used to communicate with peripheral devices attached to an HP-UX machine. In HP-UX (and all other UNIX systems), all devices are considered files, represented by device files. You can read and write data from and to the device files just like with any other file. There are two kinds of device files, *character device files* and *block device files*. Character device files represent devices which are serial in nature, like modems, disks, and terminals. They are also known as *raw device files*. Block device files represent those devices where you can read and write a block of data at one time. This type of device include disks and tape drives. Note that both character and

block device files are used for disks. Later chapters in this book explain why we use both types of device files with disks.

- Link files are special files and are used to refer to the same physical file using multiple names. There are two types of link files: hard and soft links.
- Socket files are special files used to exchange data among running processes and threads. Sockets are used either within a single system as a method of Inter-Process Communication (IPC), or between processes residing in separate systems, typically via TCP/IP. Socket files act as interfaces for a program within a system. A program opens a socket file and then other programs can connect to that socket using special library calls.
- Named pipes are special types of files that are used for exchanging data among running programs. Named pipes can be used to send and receive data. Named pipes are also called simply *pipes* or *FIFO*.

In the following sections, you learn how to determine a file type using different commands.

Using the `ls` *Command to Determine File Type*

You can use the `ls -l` or `ll` command to determine the type of a file. The first character on the left side of each line shows the *mode* of the file. The mode character tells the type of the file. Table 3.2 shows mode characters and their meanings.

Table 3.2 *Mode Character in the Output of the* `ls -l` *Command*

Mode	Description
-	The hyphen character shows an ordinary file.
c	Character device file.
b	Block device file.
l	Link file.
p	Pipe or FIFO.
d	Directory.
s	Socket file.
n	Network special file.

Look at the following output of the `ls -l` command, which shows different file types.

```
$ ls -l
total 2
-rw-rw-rw-   1 boota   users    86 Jun 21 22:33 myRegularFile
brw-rw-rw-   1 root    sys   2 0x000003 Jun 21 22:37 mybdevfile
crw-rw-rw-   1 root    sys   2 0x000003 Jun 21 22:38 mycdevfile
drwxrwxrwx   2 boota   users    96 Jun 21 19:33 mydir
prw-rw-rw-   1 boota   users     0 Jun 21 19:31 myfifo
$
```

The first file, "`myRegularFile`," is a regular file because a hyphen is the first character in this line reading from the left. The "`mybdevfile`" is a block device file because the first character reading from the left is "`b`." The file "`mycdevfile`" is a character device file because the first character from the left is "`c`." The "`mydir`" file is a directory because the first character from the left on this line is "`d`." The file "`myfifo`" is a pipe (or FIFO) because the first character from the left on this line is a "`p`."

Now you know that you can use the `ls -l` command effectively to find out the type of a file. However, this command is quite limited in what it can tell. The `file` command can provide a lot more information about files, as is explained next.

Using the `file` *Command to Determine File Type*

The UNIX `file` command is used to determine the type of a file. The `file` command looks at the first few characters of a file and tries to match them with patterns listed in the `/etc/magic` file. Using these patterns, this command can tell you the type of a particular file. See the following examples of how to determine different file types.

A TEXT FILE

The following example shows that `/etc/profile` is an ASCII text file:

```
$ file /etc/profile
/etc/profile:   ascii text
$
```

A DIRECTORY

The following command shows that `/etc` is a directory:

```
$ file /etc
/etc:           directory
$
```

AN EXECUTABLE FILE

The following command shows that `/bin/ls` is a binary executable file. It also shows that the file is dynamically linked, which means that it needs some library files at run time:

```
$ file /bin/ls
/bin/ls:    PA-RISC1.1 shared executable dynamically linked
$
```

A SHARED LIBRARY

Shared libraries are files used by different programs at run time. If you are familiar with the Microsoft Windows environment, shared library files are similar to DLL files in Windows. The following command shows that `/lib/libc.1` is a shared library file:

```
$ file /lib/libc.1
/lib/libc.1:    PA-RISC1.1 shared library -not stripped
$
```

A SHELL SCRIPT

Shell scripts are executable batch files that you can create to do different jobs. The following command shows that `abc` is a shell script file:

```
$ file abc
abc:            commands text
$
```

A NAMED PIPE

The following command shows that `myfifo` is a named pipe (FIFO) file:

```
$ file myfifo
myfifo:         fifo
$
```

A DEVICE FILE

The following command shows that `mycdevfile` is a character device type file:

```
$ file mycdevfile
mycdevfile:     character special (2/3)
$
```

Similarly, the `file` command is able to detect a number of other file types. As mentioned earlier, the `file` command uses the `/etc/magic` file to determine different file types by finding a *magic string* inside the file. A detailed discussion on magic string is beyond the scope of this book, but the man pages for `/etc/magic` offer further information on magic numbers. The `file` command is useful in situations where you want to determine the type of file before performing an operation on it. It is quite possible that your display will be garbled if you use the `cat` command on a binary file.

Study Break

Copying and Moving Files Using Wildcards and Finding the Type of a File

The general syntax of the copy (`cp`) and move (`mv`) commands is that the source filename is specified first and then the destination is specified on the command line. Create a directory with the name `impfiles` in your home directory. Copy the `/etc/hosts` file in this directory. Also, copy all files starting with "`m`" from the `/etc` directory to this directory. Now, move the `hosts` file from the `impfiles` directory to the `/tmp` directory. Using range characters `[a,e,i,o,u]`, list all files in the `/usr` directory that start with any vowel.

You can find the type of a file by using the `file` command. Try to find a shared executable file on the system by applying this command to different files.

3.8 Searching File Contents

Finding a text string in one or multiple text files is easy using the `grep` (global regular expression print) command. It does the job in a number of ways. You can search text strings in a single file or in multiple files. You can also specify additional criteria for the string, such as its occurrence at the start or the end of a line. If you are using multiple files for searching a string, `grep` also displays the name of files in which the string is found. It can also display the location in the file where the string is found.

Searching a Word

The following example shows how to determine if a particular user, Mark, exists on the system. This is done by applying the `grep` command on the `/etc/passwd` file.

```
$ grep Mark /etc/passwd
mstyle:elBY:2216:125:Mark Style,,,:/home/mstyle:/usr/bin/sh
mgany:iF5UeWQ:2259:125:Mark Gany,,,:/home/mgany:/usr/bin/sh
mbuna:tQfwUNo:2318:125:Mark Buna,,,:/home/mbuna:/usr/bin/sh
mblack:ipCg:2388:125:Mark Black,,,:/home/mblack:/usr/bin/sh
$
```

This command shows that there are four users with the name Mark on the system. Note that the search, like other HP-UX commands, is case sensitive by default. However, if you want to make a search case insensitive, you may use `grep -i` instead of `grep`. If you are interested in knowing how many times the string occurs in the file but do not wish to display the lines containing the string, use `grep -c`. You can even reverse the selection of lines by `grep -v`. In this case, all lines that *DON'T* match the string pattern are displayed.

Searching Multiple Words

If you want to search a string of multiple words, enclose the words in double quotes. For example, if you want to search "Mark Black" in `/etc/passwd`, you will use the `grep` command as follows:

```
$ grep "Mark Black" /etc/passwd
mblack:ipCg:2388:125:Mark Black,,,:/home/mblack:/usr/bin/sh
$
```

For a case-insensitive search of "Mark Black," use the following command:

```
$ grep -i "mark black" /etc/passwd
mblack:ipCg:2388:125:Mark Black,,,:/home/mblack:/usr/bin/sh
$
```

Note that without the command line option `-i`, the search will not show any matching lines.

Searching a String in Multiple Files

As I mentioned earlier, the `grep` command can be used to search multiple files for a matching string. You need to specify all filenames in which you want to search for the text string. For example, if you search for the word "root" in the `/etc/passwd` and `/etc/group` files, the following result is displayed:

```
$ grep root /etc/passwd /etc/group
/etc/passwd:root:8JgNSmFv806dA:0:3:,,,:/home/root:/sbin/sh
/etc/group:root::0:root
/etc/group:other::1:root,hpdb
/etc/group:bin::2:root,bin
$
```

The command shows that the word "root" occurs once in `/etc/passwd` and three times in the `/etc/group` file. The name of the file is listed at the start of each matching line. You can use regular expressions with the `grep` command for more sophisticated search operations. Regular expressions are discussed in Chapter 7.

3.9 Finding Files

The `find` command is used to search for a file on a system. The command requires that a search starting point and criteria for the search be specified. There are many kinds of criteria that you can use to search for files. This section contains some examples of the use of this command.

Finding Files by Filename

For example, if you want to find all files that start with "pro" in the `/etc` directory and all of its subdirectories, the command is:

```
# find /etc -name "pro*"
/etc/opt/OV/share/conf/ecs/forms/C/product
/etc/opt/OV/share/conf/ecs/forms/C/product_update
/etc/opt/OV/share/conf/OVLicense/forms/nnm/C/product.txt
/etc/opt/OV/share/conf/OVLicense/forms/nnm/C/
product_update.txt
/etc/opt/OV/share/bitmaps/C/software/process.20.pm
/etc/opt/OV/share/bitmaps/C/software/process.26.pm
/etc/opt/OV/share/bitmaps/C/software/process.32.pm
/etc/opt/OV/share/bitmaps/C/software/process.38.m
/etc/opt/OV/share/bitmaps/C/software/process.38.pm
/etc/opt/OV/share/bitmaps/C/software/process.44.pm
/etc/opt/OV/share/bitmaps/C/software/process.50.pm
/etc/opt/OV/share/bitmaps/C/software/process.38.p
/etc/opt/OV/share/bitmaps/C/software/process.16.pm
/etc/profile
/etc/protocols
/etc/X11/proxymngr
#
```

The starting point for the search is the `/etc` directory, and the criterion for the search is that the filename begins with "pro." The `find` command can also be used to find files that are newer than a particular file and to find file types and file permissions.

Finding Files by Owner

You can use the owner of the file as the criterion for the search. The following command searches all files under the `/home/boota` directory. Another criterion is that the filename must start with "my" and its owner must be "root."

```
# find /home/boota -name "my*" -user root
/home/boota/mybdevfile
/home/boota/mycdevfile
#
```

Note that here you have used two criteria for the search. Also note that you must use the `-group` keyword followed by the group name if you want to search files belonging to a particular group.

Finding Files by Size

You can specify size of a file as the search criterion. The following command searches all files under the `/etc` directory whose size is 24 blocks. Note that each block is 512 bytes long.

```
# find /etc -size 24
/etc/services
/etc/opt/resmon/lbin/monconfig.help
/etc/opt/OV/share/backgrounds/colombia.gif
/etc/opt/OV/share/www/registration/launcher/C/jovw
#
```

Also note that files that are larger or smaller than the specified size are not displayed. The following command shows all files that are 24 or more blocks long. Note that the "+" sign includes all files larger than the number specified.

```
find /etc -size +24
```

Similarly, you can use the minus sign "-" to specify files smaller than a specified number.

You can also specify the file size in bytes by appending a "c" at the end of the command argument (without a space character). The following command will display all files under the /etc directory whose size is larger than 100,000 characters:

```
find /etc -size +100000c
```

Finding Files by Access Time

You can search for files by the number of days since they were most recently accessed. The following command lists all files in the /usr/bin and /usr/sbin directories that were accessed seven days ago:

```
# find /usr/bin /usr/sbin -atime 7
/usr/bin/banner
/usr/bin/cal
/usr/bin/getconf
/usr/bin/groups
/usr/bin/logname
/usr/bin/chfn
/usr/bin/ttytype
/usr/bin/chsh
/usr/bin/nispasswd
/usr/bin/passwd
/usr/bin/yppasswd
/usr/sbin/userdel
#
```

Note that here you have used two directories as the search path. You can use multiple directories as the search path with the find command. Note that the time shows only one day. If you want to find all files accessed in the last seven days, you can specify time as -7. For example, the following command will list all files under the /etc directory that have been accessed in the last seven days:

```
find /etc -atime -7
```

Finding Files by Modification Time

The following command searches all files under the /home/boota directory that have been modified in the last twenty-four hours (0 day).

```
$ find /home/boota -mtime 0
/home/boota
/home/boota/.sh_history
/home/boota/mydir
/home/boota/myfifo
/home/boota/myRegularFile
/home/boota/mybdevfile
/home/boota/mycdevfile
$
```

Finding Files by Permissions

You are not yet familiar with file permissions. However, just keep in mind that file permissions specify which users can read, write, or execute a particular file. The `find` command can search files by permission mode. The following command searches for files in the `/usr/bin` directory that have a write permission for all system users.

```
# find /usr/bin -perm -o+w
/usr/bin/X11/xseethru
/usr/bin/sec_clientd
/usr/bin/sadp
/usr/bin/cdsclerk
/usr/bin/cue.etc/cue.dm
/usr/bin/sbvtrans
#
```

You will learn more about file permissions in Chapter 8.

Finding Files by File Type

The following command searches all files under the `/home/boota` directory that are character device files:

```
$ find /home/boota -type c
/home/boota/mycdevfile
$
```

You can also search other file types by using a single character after the `-type` keyword on the command line. These characters are listed in Table 3.3.

Table 3.3 *File Types for Searching Files*

Type	Description
c	Character device file.
b	Block device file.
f	Regular file.
d	Directory.
s	Socket file.
p	Pipe or FIFO.
l	Link file.

There are many other criteria that you can use with the `find` command. You can also perform some operations on files after searching using the `-exec` keyword at the command line. The following command finds all character device files under the `/home/boota` directory and then runs the `ll` command on these files.

```
$ find /home/boota -type c -exec ll {} \;
crw-rw-rw-   1 root       sys           2 0x000003 Jun 21 22:38 /home/
boota/mycdevfile
$
```

Note that braces `{}` indicate command arguments to the command `ll`. The `{}` are replaced at execution time by output of the `find` command. The backslash character at the end is used as an escape character for semicolon, so that the semicolon is not considered part of the `find` command.

3.10 Miscellaneous File Handling Commands

Here are some other useful command related to file handling. You will need these commands in special circumstances and will find them useful.

The Heads and The Tails

Sometimes you need to view only the first or last few lines of a text file. By default, the `head` command shows the top ten lines of a text file, and the `tail` command shows the last ten lines of a text file. For example, if you want to see the first ten lines of the `/etc/passwd` file (the file used to store usernames and passwords), the command and its output will be as follows:

```
$ head /etc/passwd
root:6XL/mkknxKSUQ:0:3::/:/sbin/sh
daemon:*:1:5::/:/sbin/sh
bin:*:2:2::/usr/bin:/sbin/sh
sys:*:3:3::/:
adm:*:4:4::/var/adm:/sbin/sh
uucp:*:5:3::/var/spool/uucppublic:/usr/lbin/uucp/uucico
lp:*:9:7::/var/spool/lp:/sbin/sh
nuucp:*:11:11::/var/spool/uucppublic:/usr/lbin/uucp/uucico
hpdb:*:27:1:ALLBASE:/:/sbin/sh
nobody:*:-2:-2::/:
$
```

Additional parameters can be used with both the `head` and `tail` commands to view as many lines of text as needed. A `tail -n 3 /etc/passwd` command will show the last three lines of the file (`tail -3 /etc/passwd` will also work). If you want to see what is being added to a text file by a process in real time, you can use `tail -f` command. This is a very useful tool to see text being added to a log file in real time. For example, you can use the following command to see any lines being added to the `/var/adm/syslog/syslog.log` file.

```
# tail -f /var/adm/syslog/syslog.log
Jun 21 22:06:04 hpux11i /usr/sbin/nfsd[1291]: nfsd 0 9  sock 4
Jun 21 22:06:04 hpux11i /usr/sbin/nfsd[1292]: nfsd 0 10  sock 4
Jun 21 22:06:04 hpux11i /usr/sbin/nfsd[1293]: nfsd 0 11  sock 4
Jun 21 22:06:04 hpux11i /usr/sbin/nfsd[1295]: nfsd 0 12  sock 4
Jun 21 22:06:04 hpux11i /usr/sbin/nfsd[1296]: nfsd 0 13  sock 4
Jun 21 22:06:04 hpux11i /usr/sbin/nfsd[1298]: nfsd 0 14  sock 4
Jun 21 22:06:04 hpux11i /usr/sbin/nfsd[1273]: nfsd 0 15  sock 4
Jun 21 22:07:16 hpux11i krsd[1711]: Delay time is 300 seconds
Jun 22 00:02:57 hpux11i su: + tb root-root
Jun 22 01:07:15 hpux11i su: + tb root-boota
```

The command will display any existing lines at the end of the file and then will remain there until it detects new data being added to the file. As soon as it finds new data added to the file, it will display it on the screen. You can use Ctrl+C to break out of this command.

Counting Characters, Words, and Lines in a Text File

Many times you want to know how many characters, words, or lines are there in a file. In the `/etc/passwd` file, for example, there is one line for every user. You can know the number of users on an HP-UX system if you count

the lines in this file. You use the `wc` (Word Count) command for this purpose. It displays lines, words, and characters in order. The following command shows information about the `/etc/profile` file:

```
$ wc /etc/profile
171 470 3280 /etc/profile
$
```

The above command shows that there are 171 lines, 470 words, and 3280 characters in the `/etc/profile` file. If you want to count only lines in a file, you can use `wc -l`. Similarly, for counting words, use `wc -w`, and for counting characters, use `wc -c`.. The following command counts the number of lines in the `/etc/passwd` file.

```
$ wc -l /etc/passwd
2414 /etc/passwd
$
```

It shows that there are 2414 lines in `/etc/passwd`, which is an indirect way to find out the number of users on this system. Note that all of these users may not be active. The number of lines shows the total user accounts that exist on the system.

Link Files

Many times you need to refer to the same file with multiple names. Creating *link files* provides a mechanism for this. You can create a link file that is not the actual file but which points to some other file to which it is linked. There are two types of links, hard and soft. Soft links may be established across file systems. The soft link is a special type of file and the first character of the `ll` command list is "`l`" for link files. To create a link, the `ln` command is used. For example, to create a hard link "`abc`" to a file `myfile`, we use the command:

```
$ ln myfile abc
$
```

To create a soft link, you use the `-s` option as follows:

```
$ ln -s myfile abc
$
```

Note that soft links are also known as *symbolic links*.

Using the `lsr` *Command*

The `lsr` command is equivalent to the `ls -R` command, which lists file in the current and all child directories recursively. The command first displays the contents of the current directory and then goes into each subdirectory and lists the files there.

■ Chapter Summary

In this chapter, you learned operations performed on files and directories. Creating, deleting, and displaying are basic operations on files. You learned how to create, delete, and change directories. Rules related to filenames were also presented in this chapter. Basic rules are that filenames are no larger than 256 characters and all alphabetic and numeric characters can be used in filenames. Now you also know that filenames in HP-UX are case sensitive. You copied and moved files from one place to another in the HP-UX directories with the `cp` and `mv` commands. The `mv` command was also used to rename files.

Operations on multiple files can be performed using the wildcards asterisk (`*`) and question mark (`?`), while square brackets can be used to specify character range. Then you learned how to determine the type of a file with the `file` command. The `grep` command was used to find a text pattern inside a text file. During this process you can also specify search criteria. Based on this criteria, you can use the `grep` command to find a string at the beginning or end of a line. You also performed a case-insensitive search. Finding files in a system with the `find` command is also covered in the chapter. You have also learned how to count characters, words, and lines of a text file with the `wc` command and how to display the first and last lines of a file using the `head` and `tail` commands, respectively. You have also seen how to create link files.

Commands presented in this chapter are very basic and important for handling files and directories on HP-UX; they require practice to get used to them. At the same time, you need to be careful about some commands that can delete your files without any warning message.

▲ CHAPTER REVIEW QUESTIONS

1. *What restrictions apply to UNIX filenames?*
2. *What is the difference between the **Home Directory** and the **Root Directory**?*
3. *Is it possible to create multiple directories with a single command line?*
4. *What is the difference between the* `cp` *and* `mv` *commands?*
5. *How do absolute path names differ from relative path names?*
6. *Write a command to list all files that start with "**my**", contain any characters in the third, fourth, and fifth positions, and end with the **e.txt** pattern.*

▲ TEST YOUR KNOWLEDGE

1. *The* `cat` *command is used to:*
   A. Create a file.
   B. Display the contents of a file.
   C. Both of the above.
   D. This is not a UNIX command, it is just the name of a pet.
2. *The maximum length of a filename may be:*
   A. 8 characters.
   B. 15 characters.
   C. 256 characters.
   D. There is no restriction on filename length in UNIX.
3. *The* `more` *command is used to:*
   A. Display the contents of text files.
   B. Display the contents of any type of file.
   C. Display all of a text file in one go.
   D. Get more help on a command.
4. *What is the function of the following command:*
   `grep "Mark Black" /etc/passwd`
   A. It searches for the string "Mark Black" in the `/etc/passwd` file.
   B. It searches word "Mark" in files `Black` and `/etc/passwd`.
   C. It is an ambiguous command with unpredictable results.

D. We can use the `search` command here instead of `grep`.

5. *Consider a directory with five files in it. The filenames are* `pg.c`, `pg1.c`, `pg2.c`, `pg3.cpp`, *and* `pg10.c`. *You use the command* `ls pg?.?`. *The files displayed are:*

   A. `pg1.c, pg2.c`

   B. `pg1.c, pg2.c, pg10.c`

   C. `pg1.c, pg2.c, pg3.cpp`

   D. `pg2.c, pg10.c`

6. *How can you determine the number of user accounts on a UNIX system?*

   A. By using the `number` command.

   B. By asking the system administrator.

   C. By counting the users one by one.

   D. By counting the lines in the `/etc/passwd` file.

7. *You are currently in* `/home/boota` *directory. Which command will bring you into the* `/etc` *directory?*

   A. `cd etc`

   B. `cd ../../etc`

   C. `cd /etc`

   D. Both options B and C.

▲ Answers to Chapter Review Questions

1. *What restrictions apply to UNIX filenames?*
   Basic restrictions are that filenames are no longer than 256 characters and that they can contain all alphabetic and numeric characters as well as some special characters.

2. *What is the difference between the* ***Home Directory*** *and the* ***Root Directory****?*
   Home directory is the default directory assigned to a user. The root directory is the `/` directory. All other directories reside inside the root directory.

3. *Is it possible to create multiple directories with a single command line?*
   Yes, you can specify multiple directory names as arguments to the `mkdir` command.

4. *What is the difference between the* `cp` *and* `mv` *commands?*
   The `cp` command copies a file, while the `mv` command moves or renames a file.

5. *How do absolute path names differ from relative path names?*
   Relative path names provide the location of a file or directory with reference to some other directory. Absolute path names provide the location of a file or directory with reference to the root directory.

6. *Write a command to list all files that start with "**my**", contain any characters in the third, fourth, and fifth position, and end with the **e.txt** pattern.*
   ls my???e.txt

▲ Answers to Test Your Knowledge

1. *The* `cat` *command is used to:*
   A. Create a file.
   B. Display the contents of a file.
   C. Both of the above.
   D. This is not a UNIX command, it is just the name of a pet.

   The `cat` *command can be used to create new files and display the contents of a file. It creates new files when you use it with the "greater than" redirection symbol (>). So option C is the correct answer.*

2. *The maximum length of a filename may be:*
   A. 8 characters.
   B. 15 characters.
   C. 256 characters.
   D. There is no restriction on filename length in UNIX.

   The maximum length of an HP-UX file is 256 characters. So option C is the correct answer.

3. *The* `more` *command is used to:*
   A. Display the contents of text files.
   B. Display the contents of any type of file.
   C. Display all of a text file in one go.
   D. Get more help on a command.

   Option A is the correct answer because the `more` *command is used to display the contents of a text file. Option B is incorrect because* `more` *is*

not suitable to display the contents of binary files. Option C is not correct because the `more` *command displays only one page of text at one time. Option D is incorrect because you use the* `man` *command to get help on other commands.*

4. *What is the function of the following command:*
   `grep "Mark Black" /etc/passwd`
   A. It searches for the string "Mark Black" in the `/etc/passwd` file.
   B. It searches for the word "Mark" in files `Black` and `/etc/passwd`.
   C. It is an ambiguous command with unpredictable results.
   D. We can use the `search` command here instead of `grep`.

   Option A is the correct answer. Option B is incorrect because the string which is enclosed in the quotation marks is the search string. Option C is incorrect because this is not an ambiguous command. Option D is incorrect because there is no `search` *command in HP-UX.*

5. *Consider a directory with five files in it. The file names are* `pg.c`, `pg1.c`, `pg2.c`, `pg3.cpp`, *and* `pg10.c`. *You use the command* `ls pg?.?`. *The files displayed are:*
   A. `pg1.c, pg2.c`
   B. `pg1.c, pg2.c, pg10.c`
   C. `pg1.c, pg2.c, pg3.cpp`
   D. `pg2.c, pg10.c`

   Option A is the correct answer because since the question mark replaces exactly one character, only `pg1.c` *and* `pg2.c` *match the criteria.*

6. *How can you determine the number of user accounts on a UNIX system?*
   A. By using the `number` command.
   B. By asking the system administrator.
   C. By counting the users one by one.
   D. By counting the lines in the `/etc/passwd` file.

   Option D is the correct answer because each line in the `/etc/passwd` *file represents one user. You can use the* `wc` *command to count lines in this file and thus find out the number of users.*

7. *You are currently in the* `/home/boota` *directory. Which command will bring you into the* `/etc` *directory?*

   A. `cd etc`

   B. `cd ../../etc`

   C. `cd /etc`

   D. Both options B and C.

   Option D is the correct answer because both of these commands will move you to the `/etc` *directory. Option B uses a relative path, while option C uses an absolute path. Option A is incorrect because it will try to move you to the* `/home/boota/etc` *directory instead of* `/etc`.