HP 9000 Series 300/800 Computers

HEWLETT PACKARD

A Beginner's Guide to HP-UX

FILE AND DIRECTORY COMMANDS

To Do This ...	Use This Command ...
Create a file named *file*.	`cat >` *file*
Set protection for specified *file*.	`chmod` *nnn file*
Set protection for specified *dir*ectory.	`chmod` *nnn dir*
Copy *fromfile* to *tofile*.	`cp` *fromfile tofile*
Copy file *frompath* to directory specified by *topath*.	`cp` *frompath topath*
Print specified *file* on printer.	`lp` *file*
List file names.	`ls`
List file names, including invisible file names.	`ls -a`
List file names and append a slash (/) to directory names.	`lsf`
Create a new directory named *dir*.	`mkdir` *dir*
Display contents of *file* on the screen.	`more` *file*
Rename file *old* to *new*.	`mv` *old new*
Move file *frompath* to directory to specified by *topath*.	`mv` *frompath topath*
Display absolute pathname of current directory.	`pwd`
Remove (delete) *file*.	`rm` *file*
Remove directory named *dir*.	`rmdir` *dir*

A Beginner's Guide to HP-UX

HP 9000 Series 300/800 Computers

HP Part Number 98594-90006

Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

Legal Notices

The information in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty. A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Printing History

The manual printing date and part number indicate its current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The manual part number will change when extensive changes are made.

Manual updates may be issued between editions to correct errors or document product changes. To ensure that you receive the updates or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details.

October 1987 ... Edition 1.

March 1988 ... Edition 2. Minor rewrite and bug fixes. Also now supports Series 800 HP-UX.

December 1988 ... Edition 3. Add security information.

September 1989 ... Edition 4. Add information on the `elm` mailer and additional task information.

Contents

Figures

Tables

1

Introduction

Welcome to the world of HP-UX, a powerful, versatile system that meets the computing needs of diverse groups of users. You can use HP-UX simply to run applications, or you can develop your own applications in its rich software development environment. In addition, HP-UX offers several powerful subsystems, such as electronic mail, windows, networking, and graphics.

This book introduces you to HP-UX—teaches you the fundamentals—so you can become productive as soon as possible. In particular, it shows you how to do these tasks:

- Get started with your first work session (logging in and out).
- Use files, the fundamental means by which HP-UX organizes information.
- Organize your files using directories.
- Protect files and directories from being accidentally deleted or altered.
- Use file name wildcard characters to enhance your productivity with file operations.
- Run HP-UX commands and learn more about them.
- Send and receive electronic mail.
- Help maintain security on your system.

Tips for Using This Guide

This guide was designed to get you "up and running" on HP-UX as quickly as possible. For example, after reading Chapter 2, you should be able to start using the system and to run a few simple commands.

Try to read each chapter in one sitting—typically, between 20 and 45 minutes are required per chapter. You also should try to read chapters 1 through 8 sequentially. When you are comfortable with the concepts in a chapter, proceed to the next chapter. Chapter 9 contains information on some additional HP-UX commands you may find useful. You can browse in this chapter any time you want to increase your HP-UX skills.

What Is a "System Administrator"?

Throughout this guide, you will see the term **system administrator.** The system administrator is a person who manages your system, taking care of such tasks as adding peripheral devices, adding new users, and doing system backups. (On some systems, the system administrator may be called the **system operator** or something similar.)

If *you* are the system administrator (for example, if you are the only user on a single-user system), then whenever this guide refers you to the system administrator, you should refer to your system administrator manuals.

Conventions

This book uses the following typographical conventions within examples:

If you see ...	It means ...
colored text	You type the text exactly as shown. For example, `more sample_file` means you should type exactly those characters.
`computer text`	Indicates text displayed by the computer system. For example, `login:` indicates a login prompt displayed by the system.
italic text	You supply the text. For example, `more` *file_name* means that you type `more` followed by a file name of your choice. Italic text also is used as annotation.
▢	You type the corresponding key on the keyboard. For example, CTRL-D means you hold down the CTRL key, and press the D key.

For More Information ...

When finished reading this book, you may want to read the other Beginner's Guides:

- *A Beginner's Guide to Text Editing*
- *A Beginner's Guide to Using Shells*

You also can read about a specific topic that interests you in other HP-UX documentation. For details on all the manuals you can get with HP-UX, see *Finding HP-UX Information.*

A Brief Word About PAM (The Personal Applications Manager)

This guide assumes you will be using either the Bourne, C, or Korn command interpreters (or **shells** as they are called in HP-UX terminology). With these shells, you run a command by typing the command's name on a **command line** and pressing the Return key.

For example, you can run the `date` command by typing:

```
$ date Return
Wed Feb 10 09:54:23 MST 1988
```

As an alternative to these command line-oriented shells, you can opt to use PAM, the Personal Applications Manager. PAM implements a visual, menu-oriented interface that uses arrow keys (or a mouse), and function keys (f1..f8).

The PAM shell is used primarily by AXE (Application Execution Environment) users. Instead of discussing PAM, this guide emphasizes command-line shells: Bourne shell, C Shell, and Korn shell. To learn more about PAM, refer to the PAM tutorial in the AXE user documentation for your system or to the *pam* entry in section 1 of the *HP-UX Reference*.

2

Getting Started: Your First Work Session

This chapter leads you through your first work session on HP-UX. You should read and do the tasks in this chapter in one sitting and in the order they appear. Do *not* read other chapters in this book until you have read this chapter.

Before working through this chapter, get your **username** and **password** from the system administrator. You will need this information in the first section, "Logging In". On some systems, you may not have a password, so your system administrator won't give you one.)

This chapter discusses the following topics:

- Logging in—identifying yourself to HP-UX so you can start a work session.
- Typing and running commands.
- Setting your password—to help ensure the security of your system.
- Logging out—exiting from HP-UX when you're finished using the system.

Logging In

To begin using HP-UX, you must log in. When you log in, HP-UX prompts you for your **username** and **password** (if you have one). On some systems, you may also have to respond to a "`TERM = (hp)`" prompt. When finished logging in, you should see a command line prompt.

Step 1: Getting a Login Prompt

To get a login prompt, press (Return) a few times until you see this message:

```
login:
```

If you don't get this prompt, consult your system administrator.

Step 2: Typing Your Username

Your username identifies you as a valid user of the system. HP-UX lets you log in only if you have a username. When you see the login prompt, type your username and press (Return). For example, if your username were `leslie`, you would type:

```
login: leslie (Return)
```

The (Back space) key does not work during log in. If you make a mistake when typing, press (Return) once or twice and HP-UX will display a new login prompt, so you can re-enter your username.

Step 3: Typing Your Password (If You Have One)

If you do *not* have a password, skip this step and go to Step 4. If you *do* have a password, you should see a password prompt after entering your username:

```
Password:
```

Your password is an "invisible" codeword that only you know. Passwords help ensure your system's security. When you see the password prompt, type your password and press (Return). To ensure that other users cannot see your password, HP-UX does *not* display the characters as you type them.

Step 4: Responding to the "TERM = (hp)" Prompt (Optional)

Every terminal or graphics display has a **terminal type** (**term-type**), which HP-UX must know to communicate with the terminal. If HP-UX doesn't set the term-type automatically, you will be prompted for it: `TERM = (hp)`

If you see this prompt, ask your system administrator for your term-type, and enter it. For example, if you use a "2392a " terminal, type:

```
TERM = (hp) 2392a    Return
```

Step 5: You're Logged In

If you completed the preceding steps correctly, HP-UX typically displays a welcome and copyright message, followed by a **command prompt.** A command prompt indicates the system is ready to accept commands. Typical prompts are **$** or **%** or something similar. (If you are using PAM (Personal Applications Manager), you'll see the PAM menu instead of a command prompt.) If you see a prompt, you can proceed to the next section "Typing Commands".

If Problems Occur ...

Listed below are some messages you might see during login and what to do if you see them. (If you see other messages, consult your system administrator.)

If you see the message ...	Do this ...
`Invalid login.` `login:`	You might have made a mistake when typing either your username or password. Repeat Steps 1-5.

`Your password has expired. Choose a new one.`
`Changing password for leslie`
`New password:`
	Change your password before logging in. Refer to "Setting Your Password".

`Maximum number of users already logged in.`	You must wait until someone logs out before you can log in.

Typing Commands

To run a command, type the command's name after the command line prompt, and press Return. To correct typing mistakes, use the Back space key.

The Command Line Prompt

When you see the command line prompt, you can begin typing commands. By default, the command line prompt is either **$** or **%**, but it can be different, depending on how your system administrator set up your account. In any case, you can locate the prompt by pressing Return several times; HP-UX displays the prompt every time you press Return:

```
$ Return
$ Return
$
```

For the purpose of clarity and consistency, examples throughout this book use the **$** *prompt.*

Running Commands

To run a command, type the command's name after the prompt and press Return. The command then will begin running. When the command finishes, the prompt reappears. For example, run the following `whoami` command now:

```
$ whoami Return
leslie
$
```

Your username is displayed here, instead of `leslie`. *Then the command line prompt reappears.*

If you make a mistake when typing a command, use the Back space key to back up and correct it.

(To make examples as clear as possible, the remainder of this book will *not* show the Return key at the end of each command line.)

To get the feel of using commands, run the `date` and `cal` commands:

```
$ date                              Displays the current date and time.
Mon Sep 28 16:56:07 MDT 1987

$ cal                               Displays an English calendar for current month.
   September 1987
 S  M Tu  W Th  F  S
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

Commands are described in more detail in Chapter 5.

A Brief Word About the Shell

When you log in, you are said to be "in" a **shell.** The shell interprets commands you type at the keyboard. HP-UX supports several different shells: the Bourne, Korn, C, and PAM shells. Your system administrator determines which shell you get when you log in.

If, after logging in, you see a Personal Applications Manager menu screen, your shell is the PAM shell. Otherwise, you can determine which shell you're "in" by typing the following command:

```
$ echo $SHELL
```

If `echo $SHELL` displays ...	Then your shell is ...
`/bin/sh`	Bourne Shell
`/bin/csh`	C Shell
`/bin/ksh`	Korn Shell

Your shell has many productivity-enhancing capabilities you may find useful after you've mastered the basics in this book. To learn more about shells, read *A Beginner's Guide to Using Shells.*

Setting Your Password

Passwords help prevent unauthorized users from logging in to the system. If you don't have a password, set one that only you know. When you've set a password, change it occasionally to ensure system security. Use the `passwd` command to set or change your password. (To learn more about system security and selecting a secure password, refer to Chapter 8 "Keeping Your System Secure".)

Choosing a New Password

Choose your new password according to these rules:

- The password must contain between six and eight characters.
- At least two characters must be letters (uppercase or lowercase).
- At least one character must be either of the following:
  - Numeric (the digits `0` through `9`).
  - Special (neither letters nor numeric—for example, `-`, `_`, or `$`).

According to these rules, the following are all valid passwords:

```
foo-bar      $money$      Number_9      @rophy
```

Also, uppercase and lowercase letters are different. Thus, these are all different passwords:

```
foo-bar      Foo-Bar      FOO-BAR
```

When you choose a password, you want to ensure that no one can guess what you chose. If someone knows your password, that person may log in and access your files. Chapter 8 "Keeping Your System Secure" offers suggestions on how to select and protect your password. These guidelines are of particular significance if you work with sensitive material.

Running the passwd Command

When you've chosen your new password, set it by running **passwd**:

```
$ passwd
```

After you've entered the **passwd** command, it leads you through these steps:

1. If you don't have a password, skip this step. If you *do* have a password, **passwd** prompts you to enter it.

   ```
   Changing password for leslie    Your username displays here.
   Old password:                   Enter your current password.
   ```

2. When you see the following prompt, enter your new password.

   ```
   New password:
   ```

3. Next, you need to re-enter the new password to verify it:

   ```
   Re-enter your new password:
   ```

You're logged in. When you log in again, use your new password.

If Problems Occur ...

If you see this ...	You might have done this ...
`Sorry.`	Typed your current password incorrectly to the "`Old password:`" prompt. Run **passwd** again.
`Password is too short - must be at least 6 characters.`	Specified a new password with less than six characters.
`Password must contain at least two alphabetic characters and at least one numeric or special character.`	Specified a new password that doesn't conform to the rules described earlier.
`They don't match; try again.`	Typed your new password differently when prompted to re-enter it.
`Too many failures - try later.`	Made too many mistakes. Try again.

Logging Out

When you're finished with your HP-UX work session, log out using the `exit` command. On some systems, you can use CTRL-D to log out. If you're using PAM, use CTRL-C. Logging out ensures that no other person can use your account when you leave. (For additional information on making your account secure, refer to Chapter 8 "Keeping Your System Secure".)

Using exit

Perhaps the most common way to log out is via the `exit` command:

```
$ exit
```

Using CTRL-D

Another commonly accepted way to log out is to hold down CTRL while pressing D (CTRL-D). On some systems, however, this may not work, and you'll see this message:

```
Use "exit" to logout.
```

Using CTRL-C (with PAM)

If you're using PAM (the Personal Applications Manager), you can exit by holding down CTRL while pressing C (CTRL-C).

The "logout" Message

When you log out, HP-UX typically displays a logout message, followed by a new login prompt. Your screen might look something like this after you log out:

```
$ exit
logout

login:
```

At this point, you or any other user can log in.

If none of the methods described above log you out, consult your system administrator.

3

Working with Files

Conceptually, a **file** is a named container in which you can store information. Files are the basic means HP-UX uses to manage information. For example, using HP-UX commands and applications, you can create files containing data pertinent to your particular needs. This chapter discusses the following topics:

- Creating files with the `cat` command.
- Listing files with the `ls` command.
- Viewing a file's contents on the screen using the `more` command.
- Printing a file using the `lp` command.
- Renaming, copying, and removing files.
- Finding out if you have permission to read, write, or alter files.

Creating and Listing Files

Using the `cat` command, you can create a file containing text. To list the names of your files, use the `ls` command.

Creating a File with cat

The examples in this chapter and in remaining chapters assume you've created a text file named `myfile`. To create the file, use the `cat` command as follows:

```
$ cat > myfile
```

After you type this command, the cursor sits on a line by itself: you can now type text into the empty file. (Press Return at the end of each line you type.) When you finish entering text, hold down CTRL and press D. The `cat` command stops and returns you to the command line prompt.

You can use the `cat` command to create your own version of `myfile`. For example, you might create the file as follows:

```
$ cat > myfile
The text I am typing will be stored in "myfile."Return
I press RETURN at the end of each line.Return
When I'm finished, I hold down the CTRL key and press D. Return
CTRL-D
$
```

Note You can also create and edit files using the `vi` text editor. To learn how to use this editor see *A Beginner's Guide to Text Editing.*

Listing Files

To verify that `cat` created `myfile`, run the `ls` command, which lists the names of your files.

```
$ ls                    The ls command lists myfile.
myfile
```

(Viewing the file's contents is discussed in the section, "Viewing and Printing Files".)

Naming Files

When you choose a file name you need to follow certain rules regarding the length of the name and the types of characters you include. If a file name begins with a dot (`.`), it is "invisible" and the `ls` command normally will *not* list it. To see invisible file names, run `ls` with the `-a` option.

Choosing a File Name

When you choose a filename, remember these rules:

- Generally, file names can contain up to 14 characters, which can be any combination of the following:
  - uppercase or lowercase letters (`A` through `Z`; `a` through `z`)
  - digits (`0` through `9`), or
  - special characters (for example, `+`, `-`, `_`, `.`).

  Based on these rules, the following are valid file names:

  ```
  money          Acct.01.87     CODE.c
  lost+found     112.3-data     foo_bar
  ```

- HP-UX interprets uppercase and lowercase letters differently in file names. Thus, the following file names all are different:

  ```
  money     Money     MoneY     MONEY
  ```

Note Some systems may be configured to accept filenames longer than 14 characters. However, before you create files with longer names check with your system administrator. If your system is not configured correctly, the longer filenames will be truncated to 14 characters, and may cause difficulties.

Invisible File Names

File names in which the first character is a dot (.) are **invisible file names,** since the `ls` command does *not* normally display them. Use invisible file names if you don't want certain files displayed when you run `ls`.

To illustrate, you have an invisible start-up file that the system runs when you log in. In HP-UX terminology, this file is called a **login script.** It is used to customize your working environment. (To learn more about login scripts, see *A Beginner's Guide to Using Shells.*)

To force `ls` to list invisible file names, including the name of your login script, run it with the `-a` option:

```
$ ls -a                          Use -a to see invisible file names.
.profile    myfile               This is the Bourne shell, so .profile is
                                 shown.
```

If your shell is ...	Then your start-up file is ...
Bourne Shell	`.profile`
C Shell	`.login`
Korn Shell	`.profile`
PAM Shell	`.environ`

Viewing and Printing Files

Using the `more` command, you can look at the contents of a text file. If your system is appropriately configured, you can print a text file using the `lp` command.

Viewing a File with more

The `more` command displays a text file's contents on the screen. For example, the following `more` command displays the contents of `myfile` (which you created in the section, "Creating and Listing Files"):

```
$ more myfile
The text I am typing will be stored in "myfile."
I press RETURN at the end of each line.
When I'm finished, I hold down the CTRL key and press D.
$
```

If the file contains more lines than are on your screen, `more` pauses when the screen is full. Try running `more` on the file `/etc/disktab`:

```
$ more /etc/disktab
# @(#) $Revision  64.5$
#
# This file contains the disktab entries for the current
# sectioning scheme.
#
# Note that the section sizes are in terms of DEV_BSIZE which
# is defined to be 1024 bytes.
#
                              .
                              .
                              .

--More--(4%)
```

The "--More--(4%)" message at the bottom of the screen means you've viewed 4% of the file thus far, and 96% of the file remains to be viewed. At this point, you can do any of the following:

- To scroll through the file a page at a time, press the space bar.
- To scroll through a line at a time, press Return.
- To quit viewing the file, press q.

Printing a File with lp

If your system is appropriately configured, you can print a text file using the `lp` (line **p**rinter) command. Before using the `lp` command, ask your system administrator this question:

- Is my system set up so I can use the `lp` command?

If not, find out why. You may have to use a command other than `lp` to get printouts. If `lp` does work on your system, ask the administrator this question:

- Where can I pick up my printout?

When you have this information, you can print `myfile` by running the `lp` command:

```
$ lp myfile
```

If the `lp` command is working properly, it should display a message indicating that it sent your file to the printer. For example:

```
request id is lp-number (1 file)
```

The *number* is a number assigned to the print job by the `lp` command. If you don't see this message, or if you get an error message consult your system administrator. If `lp` works successfully, you should get a printout with your username displayed on the first page. How long it takes to get your printout depends on how busy the system is and how fast the printer is.

For More Information ...

To find out more about printing, see Chapter 9, in the section "Getting Information About a Print Job."

Renaming, Copying, and Removing Files

To change a file's name, use the `mv` command; to make a copy of a file, use the `cp` command; to remove a file, use the `rm` command. The examples in this section assume you have created the file `myfile`, as described in "Creating and Listing Files".

Renaming Files with mv

Using the `mv` command, you can rename the file `myfile` to `foofile` as follows:

```
$ mv myfile foofile
```

To verify that `mv` renamed the file, use the `ls` command:

```
$ ls
foofile
```

To rename `foofile` back to `myfile`, type:

```
$ mv foofile myfile
$ ls                    Using ls, verify that it worked.
myfile                  It worked!
```

Caution When renaming files, take care not to rename a file using the name of another file. If you do this, the file that already has the name will be lost.

For example:

```
$ ls
afile    bfile      If you had these files ...
$ mv afile bfile    And you rename afile to bfile ...
$ ls                Look what happens ...
bfile               The previous bfile is replaced with the old afile.
```

(The `mv` command can also be used to move files to different locations on the system. This concept is discussed further in chapter 4 "Organizing Your Files: Directories".)

Copying Files with cp

Copy a file when you want to make a new version of it while still keeping the old version around. For example, to make a new copy of `myfile` named `myfile2`, type:

```
$ cp myfile myfile2
```

Now when you use the `ls` command, you will see the following:

```
$ ls
myfile      myfile2
```

Caution If you copy a file to an existing file, the existing file will be lost.

Removing (Deleting) Files with rm

If you have files that are no longer needed, you should remove (delete) them. Deleting unnecessary files leaves more room for other files on your system. For example, suppose you've finished using `myfile2`, and it is no longer needed; to remove `myfile2`, type:

```
$ rm myfile2
```

To verify that `myfile2` was removed, use `ls`:

```
$ ls
myfile
```

If You Don't Have Permission to Access Files

Files are assigned **access permissions** that control who has permission to read, alter, or remove them. If you don't have the necessary access permissions to a file, you may not be able to rename, copy, or remove it (as you have just learned to do). If this is the case, the system will display a message indicating that you can't perform the command.

The next section tells you how to find out who has permission to access files.

Finding Out Who Can Use Your Files

> Three classes of users (in various combinations) can access files: *owner, group,* and *other.* Each class may access files in various ways: read permission, write permission, and execute permission. Use the `ll` command to view file access permissions.

Access to files is restricted by classes of users. The three basic classes of users are:

- **owner**—Usually the person who created the file (for example, you).
- **group**—Several users who have been grouped together (along with you as the owner of the file) by the system administrator (for example, the members of your department).
- **other**—Any other user on the system.

Each of the above classes can access files in any of these three ways:

- **read permission**—Users with this type of permission can view the contents of a file.
- **write permission**—Users with this type of permission can change the contents of a file.
- **execute permission**—Users with this type of permission can execute (run) the file as a program by typing the filename at the command line prompt.

Using the ll Command to Display File Permissions

The `ll` (long listing) command displays the permissions for *owner, group,* and *other;* `ll` also displays the name of the file's owner and group. To see the permissions, owner name, and group name on `myfile`, for example, type the following:

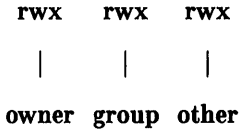
```
$ ll myfile
```

When you press Return, you should see something like this:

```
-rw-r--r--  1  leslie  users      154 Nov  4 10:18 myfile
```

The first dash in the long listing indicates that `myfile` is a file (if `myfile` were a directory, you would see a `d` in place of the dash). The next nine positions indicate read, write, and execute permissions for *owner, group,* and *other.* If a permission is not allowed, a dash appears in place of the letter.

Here is a closer view with all permissions indicated (note that the permissions are in "sets" of three):



In the example (`-rw-r--r--`), *owner* (`leslie`) has read and write permission (`rw-`); *group* (`users`) and *other* have only read permission (`r--`).

For More Information ...

To learn more about the `ll` command, refer to the *HP-UX Reference* (the `ll` command is listed under *ls* in section 1 of the *Reference*).

See Chapter 8 of this guide "Keeping Your System Secure" for information on how to change the access permissions on files and directories.

4

Organizing Your Files: Directories

After you've used your system for a while, you may start accumulating a large collection of files. With the help of directories, you can organize your files into manageable, logically related groups. For example, if you have several files for several different projects, you can create a directory for each project and store all the files for each project in the appropriate directory.

This chapter discusses the following topics:

- Organizing files in directories.
- Determining where you are in the HP-UX directory structure—using the `pwd` command.
- Using path names—to specify files and directories outside your current working directory.
- Creating your own directories using the `mkdir` command.
- Moving through the directory structure with the `cd` command.
- Moving and copying files between directories using the `mv` and `cp` commands.
- Removing empty directories with the `rmdir` command.
- Using wildcard characters—shorthand for specifying filenames.
- Finding out who can access your directories—similar to listing file permissions with the `ll` command.

What is a Directory Hierarchy?

Like files, directories are containers. But instead of text or other data, directories contain files. In addition, directories are hierachically organized; that is, a directory has a parent directory "above" and may also have subordinate child directories "below." Similarly, each child directory can contain other files and also can have child directories. Because they are hierarchically organized, directories provide a logical way to organize files.

Directories are like chambers in an oversimplified pyramid:

- Chambers contain artifacts; directories contain files.
- A chamber contains an entrance from above; a directory has a **parent directory.**
- A chamber may contain passageways to chambers below; a directory may have subordinate **child directories.**
- A pyramid has one entrance at the top; the directory structure has a **root directory** at the top.

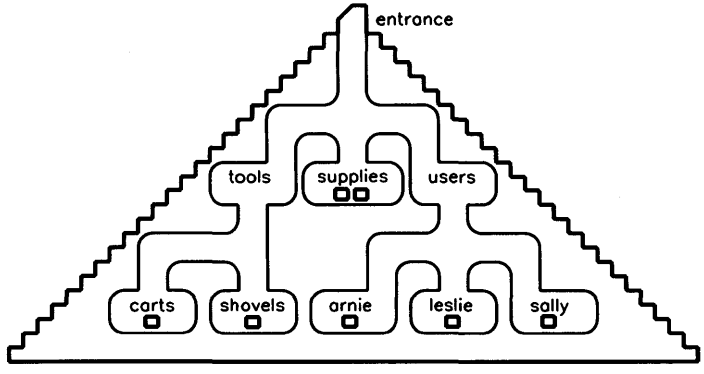


Figure 4-1. An Example Pyramid

Figure 4-2 shows the directory structure that could be used to represent the rooms and artifacts of the pyramid in Figure 4-1. (Ovals represent directories; boxes represent files.)

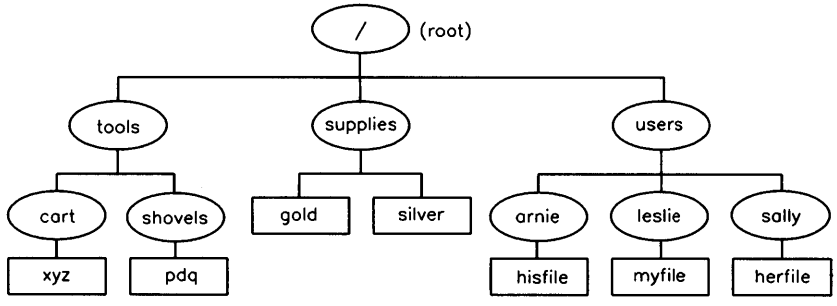


Figure 4-2. Directory Structure Mirroring the Pyramid Structure

Determining Your Position in the HP-UX Directory Hierarchy

This section discusses the HP-UX directory structure and where you are in this structure. All directories fall under the topmost **root** directory, which is denoted by a slash (/). When using HP-UX, you are always "in" a directory—your **current working directory.** And when you log in, HP-UX places you in your **home directory.**

Figure 4-3 shows the two highest levels of the HP-UX directory structure. Each directory under the root contains logically related files and directories. Directories under the root may also, in turn, have more child directories.

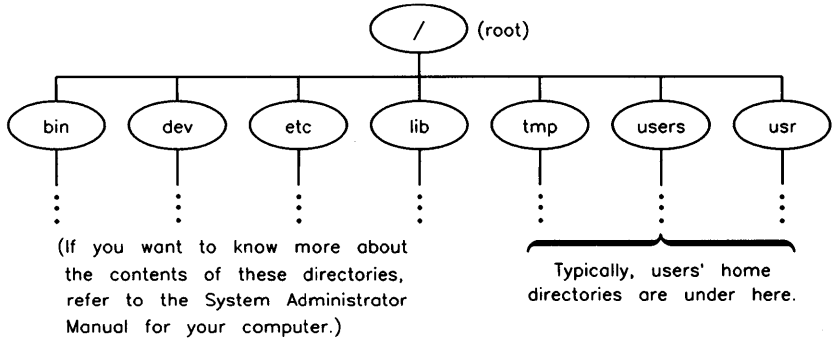


Figure 4-3. The HP-UX Directory Structure

When using HP-UX, you are always positioned "in" a directory. The directory you are "in" is known as your **current working directory.** When you log in, HP-UX always places you in a directory called your **home directory.** Logging in to your home directory is like always entering the pyramid in a particular room.

Here is an example directory hierarchy. When Leslie logs in, she finds herself in `leslie`, her home directory.

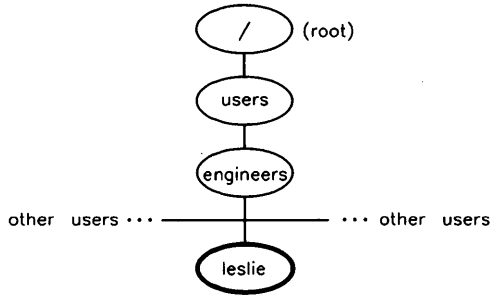


Figure 4-4. Leslie's Home Directory

Specifying Files and Directories: Absolute Path Names

> When specifying files in your current working directory, you can refer to them by their names. But when referring to directories and files outside or below your current working directory, you must use **path names.**

A path name specifies where a particular file or directory is within the directory structure. Returning to the pyramid analogy, path names are like maps showing how to get to a particular chamber or artifact in the pyramid. There are two kinds of path names: **absolute** and **relative.**

Absolute Path Names

Absolute path names specify the path to a directory or file starting from the root directory at the top of the pyramid. The root directory is represented by a slash (/); the path consists of a sequential list of directories (separated by slashes) leading to the directory or file you want to specify. The last name in the path is the directory or file you are pointing to.

To determine the absolute path to your current directory, use the **pwd** (**p**rint **w**orking **d**irectory) command. The **pwd** command displays the "path" from the root directory to your current working directory.

Here is an example using the **pwd** command:

```
$ pwd
/users/engineers/leslie
```

You used the **pwd** command to display the absolute path name of your home directory—that is, the location of your current directory, starting from the root and working down.

Figure 4-5 shows the absolute path names for various directories and files in an example directory structure:

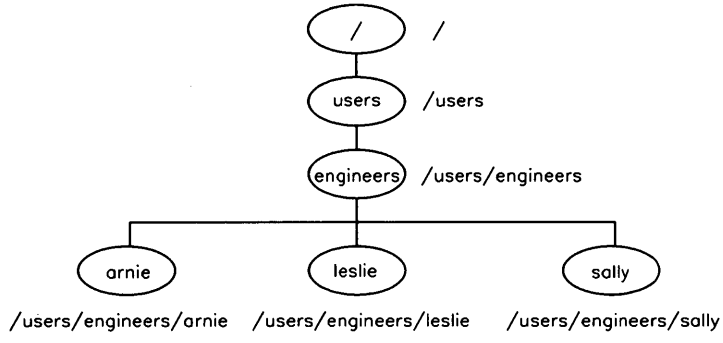


Figure 4-5. Absolute Path Names

Specifying Files and Directories: Relative Path Names

> You can use a **relative** path name as a shortcut to the location of files and directories. Relative path names specify directories and files starting from your current working directory (as opposed to the root directory).

You will frequently find it convenient to use relative path names. The following table shows some common pathname shortcuts.

Table 4-1. Examples of Relative Pathnames

This relative pathname ...	Means ...
.	the current directory.
..	the parent directory (the directory above the current directory).
../..	two directories above the current directory.
directory_name	the directory below the current directory.

As an example, suppose the current directory (as shown in Figure 4-6) is `/user/engineers/leslie`. To list the files in the directory above (which is `/user/engineers`), enter:

```
$ ls ..
arnie  leslie  sally   you get a listing of /user/engineers
```

On the other hand, to get a listing of the files in a directory immediately below your current directory, simply enter the directory name. For example, to get a listing of the files in the `project` directory, below the current directory `/user/engineers/leslie`, you would enter:

```
$ ls projects
$                    The projects directory is empty!
```

Figure 4-6 shows relative path names for various directories and files starting from the current directory, `/users/engineers/leslie`.

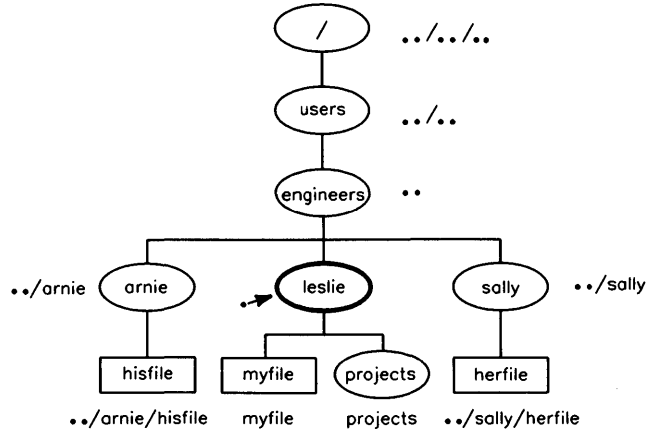


Figure 4-6.
Relative Path Names from
/users/engineers/leslie

Creating Directories

> To create a directory, use the `mkdir` command. To get a directory listing that differentiates files from directories, use the `lsf` command instead of `ls`.

After you create a directory, you can move files into it, and you can even create more directories underneath it. The `mkdir` (make directory) command creates a new directory. For example, to create a child directory (under your current working directory) named `projects`, type:

```
$ mkdir projects
```

To verify that it worked, you can use either the `ls` or `lsf` command. Both commands display the new directory, but `lsf` appends a slash (/) to the end of directory names to differentiate them from file names. For example:

```
$ ls                 List files, directories in your current working directory.
myfile  projects     It worked!
$ lsf
myfile  projects/    The lsf command appends a slash to directory names.
```

Figure 4-7 shows the resulting directory structure.

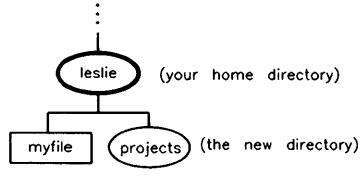


Figure 4-7. Creating the projects Directory

The general form of the `mkdir` command is as follows:

`mkdir` *new_dir_path*

where *new_dir_path* is the path name of the directory you want to create. For example, to create a new directory named `old` under the projects directory, type:

```
$ mkdir projects/old
```

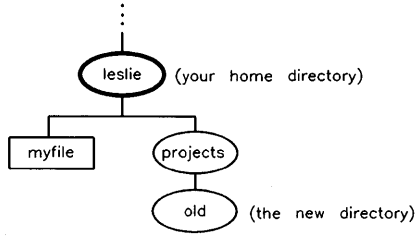


Figure 4-8.
Structure after Creating "old" under the "projects" Directory

Finally, let's create one more directory, named `new`, and verify with `lsf`:

```
$ mkdir projects/new
$ lsf projects
new/        old/          Files and directories are listed alphabetically.
```

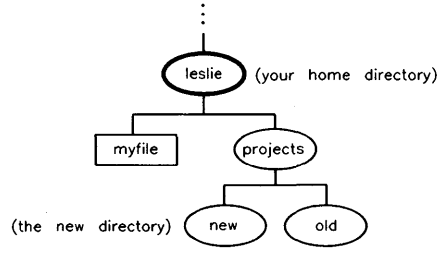


Figure 4-9. Structure after Creating "new" under "projects"

Changing Your Current Directory

Now that you've learned how to create directories under your home directory, you're ready to learn how to move into different directories, using the `cd` command.

Using the `cd` command, you can change your current working directory. For example,

```
$ cd projects
```

moves you into the directory `projects` (which you created in the section "Creating Directories"). To verify that you have, in fact, changed your current working directory, use the `pwd` command, which displays your current directory. For example, if your home directory was `/users/leslie`, then after you run the "`cd projects`" command, `pwd` would display the following:

```
$ pwd
/users/leslie/projects
```

When you're in the new directory, you can list its contents using `lsf`:

```
$ lsf                          Here are the directories you created earlier.
new/       old/
```

To move into the directory `new` under `projects`, type:

```
$ cd new
$ pwd                          Verify where you are.
/users/leslie/projects/new     It worked!
```

Now if you run `lsf`, it won't display anything because there are no files or directories under `new`:

```
$ lsf
$
```

Remember that `..` is the relative path name for the parent directory of your current working directory. So to move up one level, back to `projects`, type:

```
$ cd ..
$ pwd                          Show your current working directory.
```

```
/users/leslie/projects    It worked!
```

If you run `cd` without a path name, it returns you to your home directory as the following example illustrates:

```
$ cd
$ pwd              Are you back home?
/users/leslie      Yes!
```

Experiment with the `cd` and `pwd` commands to move around your directory structure. If you become lost, don't panic; just remember that you can run

```
$ cd
```

to return to your home directory. You can also get to any directory using its absolute pathname. For example, to change to the `projects` directory in the example hierarchy, enter:

```
cd /users/leslie/projects
```

Figure 4-10 illustrates how various `cd` commands change your current working directory. The example assumes you're starting at the directory `/users/leslie/projects`, and that your home directory is `/users/leslie`.

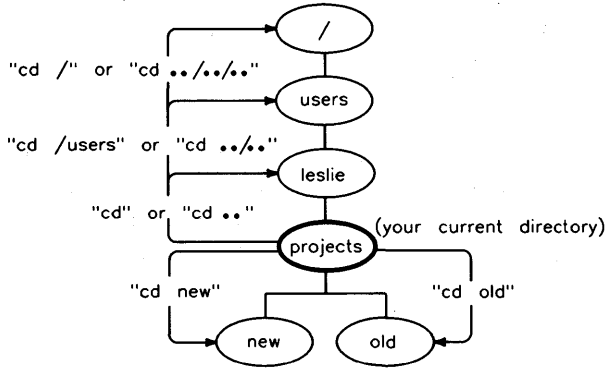


Figure 4-10. Effect of Various "cd" Commands

Moving and Copying Files between Directories

The `mv` command lets you move a file from one directory to another. With the `cp` command, you can copy a file into a different directory.

Moving Files

In addition to renaming files, the `mv` command can be used to move files from one directory to another. For example, to move `myfile` into the `projects` directory, type:

```
$ cd                          Move to your home directory first.
$ mv myfile projects
```

Now verify that it worked:

```
$ lsf                         List your current working directory.
projects/                     Where did myfile go?
$ lsf projects                Look in the projects directory.
myfile      new/      old/    There's myfile. It worked!
```

Remember that a single dot (`.`) for a path name represents your current working directory. Therefore, to move `myfile` from the `projects` directory back to your current working directory, type:

```
$ mv projects/myfile  .       Don't forget the dot.
$ lsf                         List your current working directory.
myfile      projects/         It worked; myfile is back.
$ lsf projects                List projects.
new/        old/              The file myfile isn't there anymore.
```

The general form of the `mv` command is as follows:

`mv` *from_path to_path*

where *from_path* is the file name or path name of the file you want to move, and *to_path* is the name of the path to which you are moving.

Copying Files

To copy a file into a different directory, use the `cp` command. For example, to make a copy of `myfile` named `myfile2` in the `projects` directory, type:

```
$ cp myfile projects/myfile2
```

```
$ lsf
myfile    projects/     The file myfile still exists.
$ lsf projects
myfile2   new/   old/   The copy (myfile2) is in the projects directory.
```

To make a new version of `myfile2` named `myfile3` in your current directory, type:

```
$ cp projects/myfile2 myfile3
$ lsf
myfile    myfile3    projects/
```

The general form of the `cp` command is as follows:

`cp` *from_path to_path*

where *from_path* is the file name or path name of the file you want to copy, and *to_path* is the path name of the directory or file to which you are copying.

A Caution about Moving and Copying Files

When moving or copying files, be careful not to destroy an existing file. For example, if you type the following `cp` command:

```
$ cp myfile3 projects/myfile2
```

a copy of `myfile3` is moved into `myfile2`, overwriting `myfile2`: *the previous contents of* `myfile2` *are lost.* (In this case, losing `myfile2` doesn't matter because the two files contain the same information.) As a general rule, before using `mv` or `cp`, use `ls` or `lsf` to ensure that the file to which you want to move or copy doesn't already exist.

Removing Empty Directories

When you are finished using a directory and it is no longer needed, you can remove it using the `rmdir` command. For example, if you had a directory containing 5-year-old files on a long-dead project, you might want to remove the files and the directory. Before removing a directory, you must remove its files and subdirectories.

In the pyramid analogy, removing a directory is like filling a chamber with sand. After you've removed a directory, you can no longer go into it. And before removing a directory, you must remove its files, as well as any directories under it.

For example, suppose you want to remove the `projects` directory and the files it contains. Figure 4-11 shows how this structure might look:

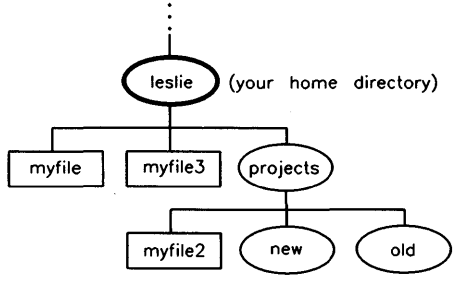


Figure 4-11. The "projects" Directory Structure

To remove this structure, run the following sequence of commands:

```
$ cd                                  Move back to your home directory

$ lsf                                 List the files and directories.
myfile    myfile3    projects/

$ rmdir projects                      Try to remove projects.
rmdir: projects not empty             It won't let you.

$ cd projects                         Change directory to projects.

$ lsf                                 List its contents.
myfile2    new/    old/

$ rm myfile2                          Remove the file myfile2.

$ lsf
new/    old/                          The file myfile2 is gone.

$ rmdir new                           Remove the directory new.  If it's
                                      empty, rmdir removes it.
$ lsf
old/                                  It worked!

$ rmdir old                           Now remove the directory old.  If it's
                                      empty, rmdir removes it.

$ lsf                                 It worked again!

$ cd                                  Now move back to your home direc-
                                      tory ...

$ rmdir projects                      And remove projects.

$ lsf                                 Verify that it worked.
myfile    myfile3                     It worked!
```

File Name Shorthand (Wildcard Characters)

Wildcard characters provide a convenient shorthand for specifying multiple file or directory names with one name. Two of the most useful wildcard characters are * and ?. The * matches any sequence (string) of characters (including no characters), and the ? matches any one character.

The * Wildcard

Suppose you have created the following files in your current working directory:

```
$ lsf
myfile    myfile2    myfile3    xorbon    yourfile
```

To list only the file names beginning with "`myfile`", type:

```
$ lsf myfile*
myfile    myfile2    myfile3
```

Even though `xorbon` and `yourfile` exist, `lsf` displays only the file names that start with `myfile`. If you wanted to list file names containing "`file`", type:

```
$ lsf *file*
myfile    myfile2    myfile3    yourfile
```

The ? Wildcard

Although you probably won't use the ? wildcard as much as *, it is still useful. For instance, if you want to list only the files that start with `myfile` and end with a single additional character, type:

```
$ lsf myfile?
myfile2   myfile3
```

The ? wildcard character matches *exactly one character*. Thus, `myfile` didn't show up in this listing because it didn't have another character at the end.

Using Wildcard Characters with mv, cp, and rm

Wildcard characters are often useful when you want to move or copy multiple files from one directory to another. For example, suppose you have two directories immediately below your current directory, named `new` and `old`, and these directories contain the following files:

```
$ lsf new
myfile    myfile2
$ lsf old
myfile3   myfile4
```

To move all the files from the directory `new` into the directory `old`, type:

```
$ mv new/* old
$ lsf new                                  The files are no longer in new.
$ lsf old                                  They are in the directory old.
myfile   myfile2   myfile3   myfile4
```

You can do a similar operation with the `cp` command. For example, to copy all the files from `old` into `new`, type:

```
$ cp old/* new
```

Similarly, you can use wildcard characters with the `rm` command.

Caution Be careful when using wildcards that you don't accidentally remove files you need.

For example, to remove all the files in the directory `new`, type:

```
$ rm new/*
$ lsf new
$              All the files are gone!
```

For More Information ...

Because wildcard characters are a feature of your shell, refer to the appropriate entry in section 1 of the *HP-UX Reference*. If you use the Bourne shell, refer to the *sh* entry, for the Korn shell, refer to *ksh*, and for the C shell, refer to *csh*.

Finding Out Who Can Use Your Directories

> Three classes of users (in various combinations) can access directories: *owner, group,* and *other.* Each user class can access directories in various ways: read permission (`r`), write permission (`w`), search permission (`x`). Search permission means that you can search the contents of the directory (for example, you can view the contents of files in the directory with `more`). Use the `ll` command to view directory permissions.

In "Finding Out Who Can Use Your Files" in Chapter 3, you learned that your files are accessible by three basic classes of users. Directories are accessible by the same three classes:

- **owner**—Usually the person who created the directory (for example, you).
- **group**—Several users who have been grouped together (along with you as the owner of the directory) by the system administrator (for example, the members of your department).
- **other**—Any other user on the same system.

Similar to permissions for files, each of the above classes may have read or write permission on a directory. Although you can't "execute" a directory, directories have "search" permission, which means that you can access the contents of files in the directory with such commands as `more`.

Table 4-2 compares permissions for directories and for files.

Table 4-2. A Comparison of Permissions for Directories and Files

This permission ...	Means this for a directory ...	Means this for a file ...
read (`r`) **permission**	Users can view the names of the files contained in that directory.	Users can view the contents of the file.
write (`w`) **permission**	Users can create, rename, or remove files contained in that directory.	Users can change the contents of the file.
execute/search (`x`) **permission**	Users can search for (access) files contained in that directory. For example, with search permission, users can use `more` to view the contents of files in the directory.	Users can execute (run) the file as a program by typing the filename at the command line prompt.

Using the ll Command to Display Directory Permissions

> You can display access permissions for a directory, as for a file, with the `ll` command. To display permissions for a specific directory, use the `ll` command with the `-d` option.

To display permissions showing *owner, group,* and *other* for a specific directory, use the `ll` command with the `-d` option. For example to see the permissions on the `projects` directory below the current directory, type the following:

```
$ ll -d projects
```

Follow the `ll` *command with a* `-d` *and the directory name.*

When you press (Return), you should see something like this:

```
drwxr-x---  1  leslie  users      1032 Nov  28 12:38 projects
```

The first character (`d`) in the long listing above indicates that `projects` is a directory. The next nine positions (three sets of three) indicate read (`r`), write (`w`), and search (`x`) permissions for *owner, group,* and *other.* If a permission is not allowed, a dash appears in place of the letter. Here is a closer view with all positions indicated:

```
d      rwx    rwx    rwx
|      |      |      |
directory owner group other
```

Now let's return to the example (`drwxr-x---`): *owner* (`leslie`) has read, write, and search permission (`rwx`); *group* (`users`) has read and search permission (`r-x`); *other* has no access (`---`) to the `projects` directory.

For More Information ...

To learn more about the `ll` command, refer to the *HP-UX Reference* (the `ll` command is listed under *ls* in section 1 of the *Reference*).

See Chapter 8 of this guide "Keeping Your System Secure" for information on how to change the access permissions on files and directories.

5

Command Syntax and On-Line Help

HP-UX has many powerful, versatile, useful commands. This brief chapter provides some background information that will help you to use commands more effectively.

More specifically, this chapter discusses the following topics:

- Understanding command syntax—command options and arguments.
- Displaying on-line *HP-UX Reference* entries (man-pages), if they are installed on your system.

Understanding Command Syntax

HP-UX provides a wealth of powerful, versatile commands that can do a wide variety of computing tasks. Before delving into these commands, though, you should have a general understanding of command syntax.

Most of the commands you've used thus far have been simple in syntax; that is, they've been either a command without any arguments (`whoami`), or a command whose only argument is a file name (`mkdir projects`). In actuality, commands can be more complex, having additional options, arguments, or both. **Options** change a command's behavior. (For example, in Chapter 3, you used the `-a` option to change the behavior of the `ls` command so you could list invisible file names.) **Arguments** provide additional information needed by the command. In general, command options are preceded by a dash (`-`).

Examples Using Options

When used without any options, the `rm` command removes a file without verifying whether you really want to remove it. Suppose, for example, your current working directory contains these files: `myfile`, `myfile1`, `myfile2`, `myfile3`, and `myfile4`. You could remove all these files by typing this command:

```
$ rm my*
$                    All the files are removed, no questions asked.
```

If you want `rm` to prompt you for verification *before* removing each file, use the `-i` option:

```
$ rm -i my*
myfile:  ? y     Type y to remove each successive file; n to leave it alone.
myfile1: ? y
myfile2: ? y
myfile3: ? y
myfile4: ? n     You don't want to remove this file, after all.
$ ls
myfile4          It worked ... rm did not remove myfile4.
```

Examples Using Arguments

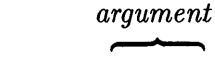
As you learned in "Typing Commands" in Chapter 2, the `cal` command displays an English calendar for the current month. With command arguments, you can specify which calendar month and year to display. For example, to display a calendar for September, 1752, type the `cal` command as follows:

```
$ cal 9 1752
   September 1752
 S  M Tu  W Th  F  S
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

(This is not an error. When England switched from a Julian to a Gregorian calendar in September, 1752, 11 days were removed from this month.)

Enclosing Arguments in Apostrophes

When an argument contains embedded blanks, you must enclose it between apostrophes ('word1 word2') . For example, the following `grep` command displays all lines in `myfile` containing "`I am`":




```
$ grep 'I am' myfile
The text I am typing will be stored in "myfile."
```

Running Multiple Commands on the Same Command Line

Occasionally, you may find it useful to run two or more commands on the same command line. To do so, separate the commands with a semicolon, as illustrated below:

```
$ whoami ; date
leslie                          Output from whoami
Fri Oct  2 15:51:57 MDT 1987    Output from date
```

Accessing On-Line Command Help: man-pages

HP-UX commands are documented thoroughly in the *HP-UX Reference.* In addition, HP-UX can also store *HP-UX Reference* entries (commonly known as **man-pages**) on disk. If your system has these "on-line" man-pages, you can use the `man` command to display them on the screen.

The `man` (**man**ual) command displays a command's syntax plus a detailed description of the command and its options and arguments (if any). Also, `man` may display examples of command usage and provide other information such as system files used, related commands, diagnostics, possible problems (bugs).

For example, you can use `man` to learn more about the `man` command itself:

```
$ man man
  MAN(1)                                                    MAN(1)

  NAME
       man - find manual information by  keywords;
       print out the manual

  SYNOPSIS
       man -k keyword ...
       man -f file ...
       man [ - ] [ section ] title ...

  DESCRIPTION
       Man is a program that gives information from the
       HP-UX Reference  Manual ...

  --More--(11%)
```

The message at the bottom of the screen means there's more of the file for you to see (on some systems, you'll see `--More--(11%)`, which means you've viewed 11% of the file, and 89% remains). At this point, you can do any of the following:

- Step through the file a page at a time by pressing the space bar.
- Scroll through a line at a time by pressing Return.
- Quit viewing the reference page by pressing q.

(You can't move backward through the file; only forward.)

Note: It may take anywhere from a few seconds to a few minutes for the **man** command to display information.

For details on the command's syntax, refer to the "SYNOPSIS" section. The above example shows, for instance, that the **man** command actually has three different syntaxes:

```
man -k keyword ...
man -f file ...
man [ - ] [section] title ...
```

The brackets, [ ], indicate that the enclosed parameter is optional. For example, when you ran the **man** command, you used the third syntax, without the optional parameters:

```
man [ - ] [section] title ...
```

```
$ man man
```

You should experiment with using the **man** command to learn more about the various commands we've discussed thus far; for instance, to learn more about the `ls` command, type:

```
$ man ls
```

To learn more about the `cp` command, type:

```
$ man cp
```

6

How the Shell Interprets Commands

By learning about special features of shells, you can become more proficient with HP-UX. This chapter introduces you to some of the useful features of shells.

Specifically, this chapter discusses the following topics:

- What a process is, and how it is created.
- How commands handle standard input, standard output, and standard error.
- Redirecting command output to a file or device such as a printer, instead of to the screen.
- Redirecting command input to come from a file, instead of the keyboard.
- Combining output and input redirection in one command line.
- Redirecting standard error from the screen to a file or device.
- Connecting two or more processes with pipes, so the standard output of one process becomes the standard input to the next process.

How the Shell Interacts With Processes

When you log in, you are said to be "in" a **shell.** The shell interprets commands you type at the keyboard. After the shell interprets a command line, HP-UX loads the named program (command) into memory and begins running the program. When a program is loaded and running, it is called a **process.** HP-UX assigns every process a unique number, known as a **process identifier** (**PID**).

How to Create a Process

When you log in, HP-UX starts your shell. During login, HP-UX copies the shell program from system disk into memory. When it is in memory, the shell program begins executing, and it becomes a process that lasts until you log out.

Similarly, the commands you type create processes. After you type a command line, the following events take place:

1. The shell interprets the command line and searches the disk until it finds the requested program.
2. The shell asks HP-UX to run the program; then control transfers from the shell to HP-UX.
3. HP-UX copies the specified program from a disk file into memory. When the program resides in memory, it begins executing—and a process is created.
4. When the program finishes executing, control transfers back to the shell, and the process disappears.

Note: An important difference exists between the terms program and process. **Program** refers to the file stored on the disk. **Process** refers to the copied program that is actively executing in memory.

What Are Process Identifiers?

HP-UX assigns every process a unique number called a **process identifier** or **PID.** HP-UX manages system resources and determines when each process runs. The PID identifies each individual process, thus allowing HP-UX to track all the processes that are running at any given time.

For More Information ...

Your shell has many powerful, productivity-enhancing capabilities you may find useful after you've mastered the basics in this book. To learn more about processes, shells in general, and the capabilities of your particular shell, read *A Beginner's Guide to Using Shells.*

Overview: Understanding Standard Input, Standard Output, and Standard Error

> When a process begins executing, HP-UX automatically opens three files for the process: "standard input", "standard output", and "standard error." Standard input usually is read from the keyboard; standard output and standard error usually are written to the terminal screen.

Each process opens three standard files: standard input (`stdin`), standard output (`stdout`), and standard error (`stderr`). Programs use these files as follows:

- **Standard input** is the place from which the program expects to read its input. By default, processes read `stdin` from the keyboard.
- **Standard output** is the place the program writes its output. By default, processes write `stdout` to the terminal screen.
- **Standard error** is the place the program writes its error messages. By default, processes write `stderr` to the terminal screen.

Figure 6-1 illustrates the relationship of these files to the process.

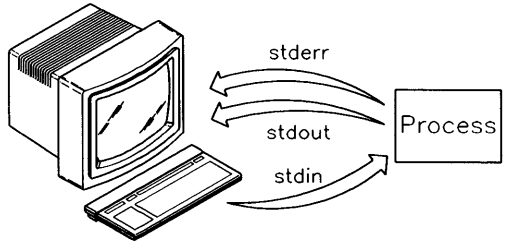


Figure 6-1.
Standard Input, Standard Output, and Standard Error

Examples Using the Standard Files

The following three examples illustrate standard output, standard input, and standard error, respectively:

```
$ whoami        The whoami command begins the process.
terry           Standard output is displayed on the screen.
$ _             Control returns to the shell.
```

The above example illustrates standard output. In this example, the `whoami` command uses standard output to display the username of the person typing the command. The prompt returns, indicating that the shell is ready for another command.

```
$ sort          The sort command uses standard input.
duffy           Enter standard input at the keyboard.
muffy
daffy
CTRL-D          End of standard input.
daffy           Standard output is displayed on the screen.
duffy
muffy
$ _             Control returns to the shell.
```

The above example uses the `sort` command to sort text typed at the keyboard. Typing CTRL-D ends standard input. The standard output is displayed on the terminal screen.

```
$ mroe memo          The process begins.
mroe: not found      Standard error is displayed on the screen.
$ _                  Control returns to the shell.
```

The standard error example, above, illustrates what happens if you misspell a command. The typing error causes the shell to generate an error message that is sent to standard error. Again, the prompt returns, indicating that the shell is ready for another command.

Writing Standard Output to a File

The shell lets you **redirect** the standard output of a process from the screen (the default) to a file. Redirecting output lets you store the text generated by a command into a file; it's also a convenient way to select which files or devices (such as printers) a program uses. To redirect a process's output, separate the command and the output file name with a greater-than sign (>); to append the output to an existing file, use two greater-than signs (>>).

Redirecting Standard Output

You can use output redirection with any command that writes its output to `stdout` (your screen). You can redirect output either to a new file or to an existing file. To redirect output, use a greater-than sign (>). The word following the sign identifies the file name where the `stdout` data is to be written. If the file exists, its previous contents are lost. If the file does not exist, it is created. In its simplest form, the command syntax is as follows:

command > *outfile*

where *command* is the command whose output is redirected, and *outfile* is the name of the file to which the process writes its standard output.

The example below shows output redirection using the `who` command, which displays a list of users currently logged in to the system. Instead of displaying the users on the terminal screen, the output is redirected to the file `whoison`. The `cat` command lists the contents of the `whoison` file, showing that the output redirection was successful:

```
$ who > whoison                                  Redirect output to whoison.
$ cat whoison                                    Display contents of whoison.
pat         console      Oct  9 08:50
terry       tty01        Oct  9 11:57
kim         tty02        Oct  9 08:13
$
```

Figure 6-2 illustrates where `stdin`, `stdout`, and `stderr` go when output is redirected to a file.

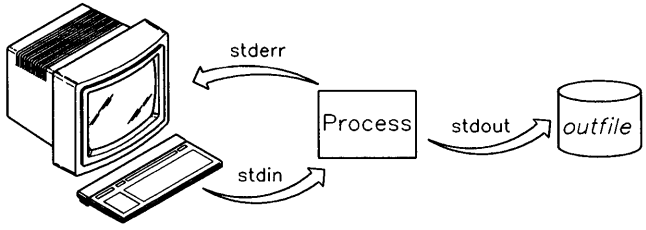


Figure 6-2.
Standard Input, Output, and Error When Output Is Redirected

Appending Output to a File

Output redirection that appends to an existing file uses two greater-than signs (>>). The word following the redirection sign identifies the file to which `stdout` data is appended. If the file exists, the new data is appended to the end of the file. If the file does not exist, it is created. The command syntax is:

command >> *outfile*

where *command* is the command whose output is redirected, and *outfile* is the name of the file to which the process appends the standard output.

The next example executes the `date` command with the output redirected to append to the `whoison` file:

```
$ date >> whoison                          Append output to whoison.
$ cat whoison                              Display contents of whoison.
pat         console      Oct  9 08:50
terry       tty01        Oct  9 11:57
kim         tty02        Oct  9 08:13
Fri Oct  9 13:20:16 MDT 1987
$
```

Using Files for Standard Input

> The shell lets you redirect the standard input of a process so that input is read from a file instead of from the keyboard. To redirect a process's input, separate the command and the input file name with a less-than sign (<).

Your shell can redirect the data flow into a program so that the keyboard is not used. Input that is normally typed at the keyboard can be redirected to be read from a file. You can use input redirection with any command that accepts input from **stdin** (your keyboard). You cannot use input redirection with commands, such as **who**, that do not accept input from **stdin**.

To redirect input, use a less-than sign (<). The word following the sign identifies the file from which the **stdin** data is read. The file must exist for the redirection to succeed. In its simplest form, the command syntax is as follows:

command < *infile*

where *command* is the command whose input is redirected, and *infile* is the name of the file from which the process reads standard input.

The following example illustrates input redirection. First, standard output from the **who** command is redirected to a file named **newwhoison**. Second, the **cat** command displays the contents of **newwhoison**. Finally, standard input for the **wc** (word count) command is redirected to come from the **newwhoison** file:

```
$ who > newwhoison                          Redirect output to newwhoison
$ cat newwhoison                            Display contents of newwhoison
pat         console     Oct  9 08:50
terry       tty01       Oct  9 11:57
kim         tty02       Oct  9 08:13
kelly       tty04       Oct  9 10:04
$ wc -l < newwhoison                        Redirect input from newwhoison
           4
$ _
```

In the preceding example, the `wc` command with the `-l` option counts the number of lines in the input file. Because input is redirected from `newwhoison`, this number equals the number of users logged in to the system when the `who` command was executed.

Figure 6-3 illustrates where `stdin`, `stdout`, and `stderr` are directed when input is redirected from a file.

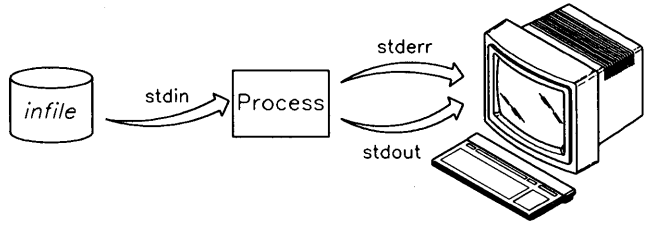


Figure 6-3.
Standard Input, Output, and Error When Input Is Redirected

Writing Both Standard Input and Standard Output to Files

You can use one command to redirect both standard input and standard output. *Do not use the same file name for standard input and standard output. When input and output operations use the same file, the original contents of the input file are lost.*

Using the Default Standard Input and Standard Output

The following example uses the `sort` command to sort text typed at the keyboard. Typing CTRL-D ends standard input. The standard output displays on the terminal screen as follows:

```
$ sort
muffy
happy
bumpy
CTRL-D                    End of standard input.
bumpy
happy
muffy                     End of standard output.
$ _
```

Redirecting Standard Input

In the following example, input is redirected:

```
$ cat socks               Display contents of socks.
polka dot
argyle
plaid
$ sort < socks            Redirect input from socks and sort the
                          contents.
argyle
plaid
polka dot
$ _
```

In the preceding example, the `sort` command uses a file named `socks` as input. As with the first example, the standard output displays on the terminal screen.

Using Both Standard Input and Standard Output Redirection

The next example combines both input and output redirection:

```
$ sort < socks > sortsocks      Use both input and output redirection.
$ cat sortsocks                 Display contents of sortsocks.
argyle
plaid
polka dot
$ _
```

In this example, the `sort` command reads input from the `socks` file and writes output to the `sortsocks` file; thus, standard output (unlike the first two examples) does not display on your screen.

Figure 6-4 illustrates where `stdin`, `stdout`, and `stderr` are directed when both output and input are redirected from and to files.

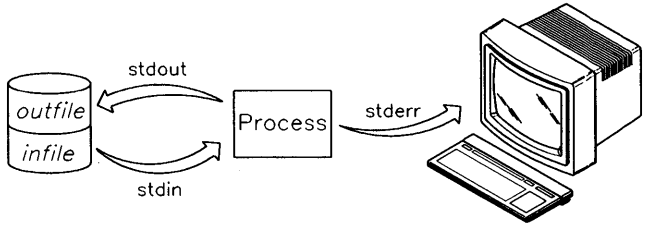


Figure 6-4. Redirecting Both Input and Output

Writing Standard Error to a File

Most programs write error messages to the default standard error (the screen). But the shell lets you redirect the standard error of a process from the screen to a file. Redirection symbols and syntax vary among the shells.

Your shell can redirect error messages to a file so the terminal screen is not used. Redirecting standard error is a convenient way to write error messages to a file. In the Bourne Shell and Korn Shell, the standard error file is associated with file descriptor digit 2. The **file descriptor digit** tells the shell what standard file is referenced:

- File descriptor 0 is associated with standard input.
- File descriptor 1 is associated with standard output.
- File descriptor 2 is associated with standard error.

To redirect a process's standard error, separate the command and the error file name with the appropriate file descriptor digit (2) and a greater-than sign (2>). In its simplest form, syntax for standard error redirection is as follows:

command 2> *errorfile*

where *command* is the command whose `stderr` is redirected, and *errorfile* is the name of the file to which the process writes its standard error.

Note: C Shell syntax is:

(*command* > *outfile*) >& *errorfile*

Usually the difference between standard output and standard error is hard to see. For example, if you ask the `ls` command to display a file that does not exist, the command generates an error message that is written to your terminal screen. Because both standard output and standard error use the terminal screen, nothing visibly distinguishes between the two standard files.

In the example below, "**nonesuch not found**" is an error message written to standard error. By redirecting the standard error, the error message is redirected to the file **noneerror** and does not display on the terminal screen:

```
$ ls nonesuch
nonesuch not found              Standard error message.
$ ls nonesuch 2> noneerror      Redirect stderr.
$ cat noneerror                 Display contents of noneerror.
nonesuch not found
$ _
```

In the example, standard error is redirected to a file, but standard output still prints on your terminal screen. This example is not typical of how to use standard error redirection. More typically, you redirect both **stdout** and **stderr**, and check the contents of the files later.

The following section describes how to redirect both standard error and standard output.

Redirecting Both Standard Error and Standard Output

In the preceding section, you learned how to redirect standard error to a file and still print standard output on the terminal screen. Usually, however, you redirect *both* standard output and standard error. Standard output and standard error can be written to different files or to the same file.

To redirect `stderr` and `stdout` to *different* files, use the following syntax:

command `>` *outfile* `2>` *errorfile*

where *command* is the command whose `stdout` and `stderr` are redirected, *outfile* is the name of the file to which the process writes `stdout`, and *errorfile* is the name of the file where the process writes `stderr`.

Note: C Shell syntax to redirect output and error to different files is:

(*command* `>` *outfile*) `>&` *errorfile*

To redirect `stderr` and `stdout` to the *same* file, use the following syntax:

command `1>` *outfile* `2>&1`

where *command* is the command whose `stdout` and `stderr` are redirected, and *outfile* is the name of the file to which the process writes `stdout`. The `2>&1` tells the shell to write standard error (file descriptor 2) in the file associated with standard output (`>&1`).

Note: C Shell syntax to redirect output and error to the same file is:

command `>&` *outfile*

Figure 6-5 illustrates where **stdin**, **stdout**, and **stderr** are directed when output and error are redirected to a file.

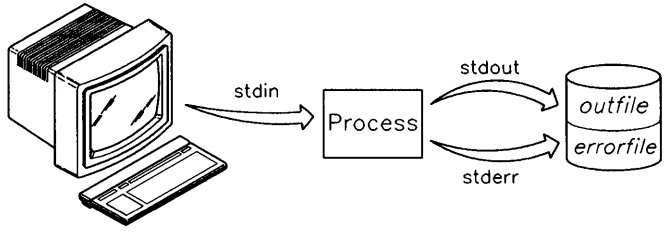


Figure 6-5.
Standard Input, Output, and Error
When Output and Error Are Redirected

Using the Output of One Command as Input to Another: Pipes

> The shell lets you connect two or more processes, so the standard output of one process is used as the standard input to another process. The connection that joins the processes is a **pipe;** using pipes within a command line is a **pipeline.** To pipe the output of one process into another, separate the commands with a vertical bar (|).

In previous sections, standard output and standard input were redirected to and from files. Using a pipe, output from one command is sent directly to another command as input. A pipe can link any two programs provided the first program writes its output to `stdout` and the second program reads its input from `stdin`. The general syntax for a pipe is as follows:

command1 | *command2*

where *command1* is the command whose standard output is redirected or piped to another command, and *command2* is the command whose standard input reads the previous command's output. You can combine two or more commands into a single pipeline. Each successive command has its output piped as input into the next command on the command line:

command1 | *command2* | ... | *commandN*

You can use pipes whenever one command needs the output of another command. In the following example, output from the `who` command is stored in the file `newwhoison`. Then, the `newwhoison` file is used as input to the `wc` command:

```
$ who > newwhoison        Redirect output of who to file newwhoison.
$ wc -l < newwhoison      File newwhoison is input to wc command.
   3                      Sample result.
```

With a pipeline, these two commands become one:

```
$ who | wc -l
   3
```

As this example illustrates, using pipes bypasses the need for temporary intermediate files. Instead, the standard output from the first command is sent directly to the second command as its standard input.

Using the tee Command with Pipes

The `tee` command lets you divert a copy of the data passing between commands to a file without changing how the pipeline functions. The example below uses the `who` command to determine who is on the system. In the example (which is further illustrated in Figure 6-6), the output from `who` is piped into the `tee` command, which saves a copy of the output in the file `savewho`, and passes the unchanged output to the `wc` command:

```
$ who | tee savewho | wc -l
      3
$ cat savewho
pat         console      Oct  9 08:50
terry       tty01        Oct  9 11:57
kelly       tty04        Oct  9 10:04
$ _
```

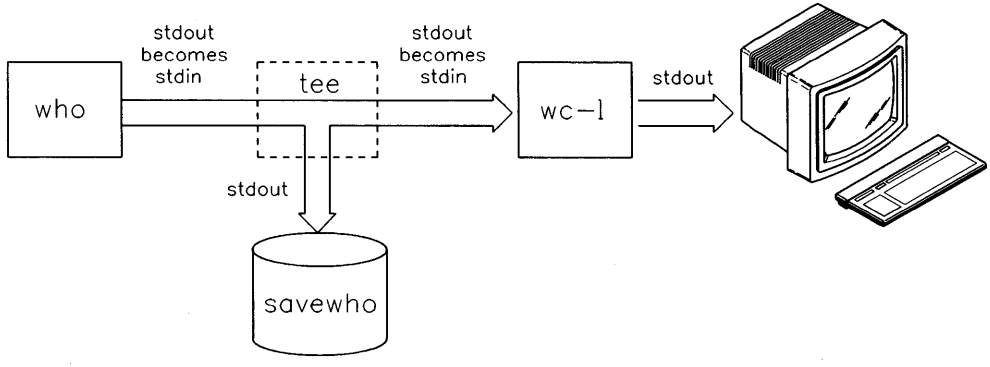


Figure 6-6. Standard Input and Output with Pipes and tee Command

For More Information ...

HP-UX provides **filter programs** that are useful in pipelines. These programs accept text as input, transform the text in some way, and produce text as output. Filter commands include `adjust`, `awd`, `cat`, `cut`, `grep`, `head`, `more`, `pr`, `rev`, `sed`, `sort`, `spell`, and `tail`. For information on these commands, refer to the *HP-UX Reference* (section 1).

7

Sending and Receiving Mail

If you are on a multi-user system, you can send mail messages to other users on your system using the `elm` mailer. And if your system is connected to a network, such as a local area network (LAN), you may also be able to send mail messages to users on other systems.

This chapter does *not* describe everything there is to know about using the `elm` mail. But it does describe enough for you to send and read mail messages.

Specifically, this chapter discusses the following topics:

- Sending mail to users on your system
- Reading your mail
- Sending mail to users on other systems
- Saving and deleting mail messages
- Customizing `elm`
- Using common mail commands

Getting Started with the Elm Mailer

Using the `elm` mailer, you can send mail messages to other users on your system. To bring up the `elm` mailer, enter

```
$ elm
```

You see a display similar to the following:

```
   Mailbox is '/usr/mail/leslie' with 3 messages. [Elm revision: X.X]


 N  1   Apr 2   Robert   (24)   Meeting Tommorrow
    2   Apr 2   Lynn     (154)  More Software Requests
    3   Apr 3   Patrick  (78)   Hi there




You can use any of the following commands by pressing the  first character;
D)elete or U)ndelete mail, M)ail a message, R)eply or F)orward mail, Q)uit.
To read a message, press  <return>.  j = move down, k = move up, ? = help


Command: _


 Read    Mail    Reply   Save              Delete  Undelete  Print   Quit
  Msg     Msg   to Msg    Msg               Msg     Msg       Msg     Elm
```

You can enter an `elm` command in two ways:

- Type the first letter (uppercase or lowercase) of the command.
- Press the function key that corresponds to the command. (You can execute the command menu choices, `Read Msg`, `Mail Msg`, etc. that appear at the bottom of the screen by pressing function keys f1 through f8 at the top of your keyboard.)

The examples in this guide use the first method, typing the first letter of the command.

Leaving Elm

Now that you're in `elm`, practice leaving it. Enter

```
Command: q          Keep mail in incoming mailbox ? (y/n) y
```

Respond `y` to the prompt or press Return.

Any messages in the incoming mailbox remain there, and the shell prompt returns. (If you answer `n` to the prompt, messages are stored in an alternate mailbox; the default is *homedirectory*`/mbox`.)

Sending Mail to Other Users on Your System

One of the easiest ways to learn how to send a mail message is to send one to yourself. If you're not already in **elm**, enter:

```
$ elm
```

To mail a message, type:

```
Command: m
```

Elm responds with a prompt requesting the **mail address** of the recipient.

```
Send the message to: leslie    Enter your own user name.
```

Elm then responds with a **subject line** prompt:

```
Subject of message:    Type the subject line for the message.
```

For the message we're about to type, you might enter a subject line like:

```
Subject of Message: Important Message to Myself
```

After entering the subject line, press Return. Elm responds with a prompt for the carbon copies:

```
Copies To:    Since you're sending a message to yourself,
              you don't want to send any additional copies
              so press Return.
```

On most systems, **elm** brings up the **vi** editor. (Some systems may be configured to bring up a different editor).

If you're in **vi**, press **i** to enter insert mode, and then begin typing the message. At the end of each line, press Return.

```
This is a mail message sent to myself.   Type the message.
Does it work?
We'll soon see.

Goodbye,
Leslie
```

To exit the **vi** editor and save your message, press esc and then enter:

```
:wq
```

(For more information on the `vi` editor see *A Beginner's Guide to Text Editing.*)

After you exit the editor, you'll see the following message on the screen.

```
Please choose one of the following options by the first character:
E)dit message, edit H)eaders, S)end it, or F)orget it. s
```

To mail your message, enter

```
s
```

After you've sent the mail message, the `elm` main screen reappears, and the message "`Mail Sent`" is displayed.

It might take a few minutes for the system to deliver the message. To learn how to read the message you just sent to yourself, see "Reading Your Mail".

Sending a Message to Multiple Recipients

To send a message to multiple users, specify each user's name next to the `elm` prompt for a recipient.

```
Send the message to: JaneS PaulaW JasonL
```

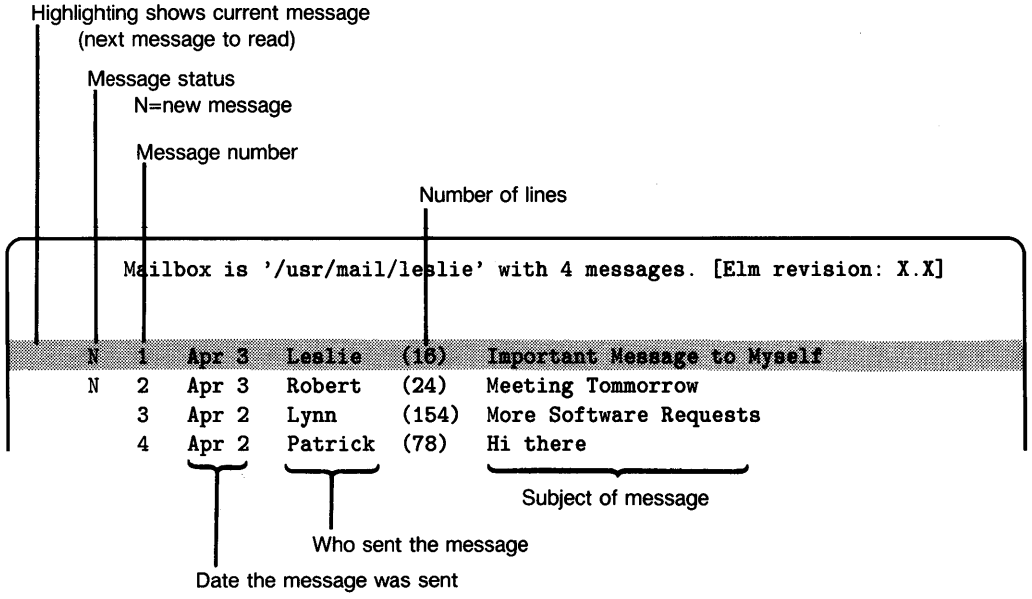
Reading Your Mail

To read your mail, bring up the **elm** mailer. If you **do** have mail, **elm** displays a list of mail messages. You can read the **current** message or pick a specific message to read.

To determine whether you have any mail, type:

```
$ elm
```

The **elm** screen appears. If you have messages, **elm** lists the number of messages. You'll see a display like the following:



To read the **current** message (that is, the message highlighted in inverse video) press (Return). (On some systems the current message may be indicated by a > to the left of the message.)

The following example shows the output from reading message 1 - the message to yourself.

```
Message 1/4 from Leslie Pendergrast

Subject: Important Message to Myself
To: leslie
Date: Mon, 10 Jul 89 16:26:45 PDT
Cc:

This is a mail message sent to myself.
Does it work?
We'll soon see.

Goodbye,
Leslie
```

To return to the `elm` main screen, press Return.

More Tips on Reading Messages

You can select a message to read as follows:

- Type `j` to advance to the next message, `k` to move to the previous one. Press Return to read the message.
- To read a specific message, type the number of the message and press Return.

Displaying Message Headers

Only ten message headers are listed on the screen at one time. If you have more than ten messages you can display them as follows:

- To see the next page of message headers, press `+`.
- To see the previous page, press `-`.
- To move to the first message in the list, press `=`.
- To move to the last message in the list, press `*`.

Sending Mail to Users on Other Systems

If your system is connected to other systems over a LAN (local area network), UUCP or other similar networking facility, you may be able to send mail over the LAN to users on other systems.

Node Names

Every system connected over the LAN has a unique **node name** (also known as a **host name**). When sending mail to each other over LAN, systems must know each others' node names. Figure 7-1 shows an example LAN with four systems connected. The node names are **research, develop, market,** and **sell.**

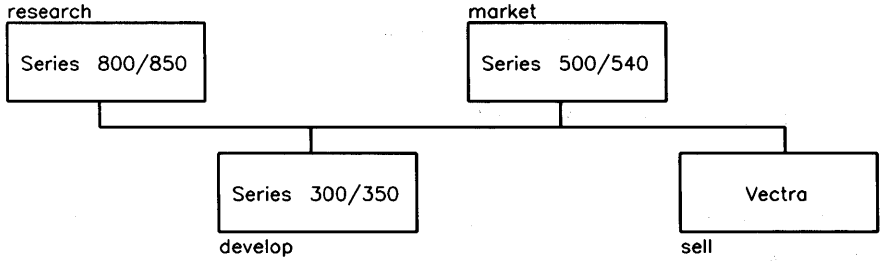


Figure 7-1. Example LAN and Node Names

To determine your system's node name, use the **hostname** command.

```
$hostname
market        Your hostname is displayed.
```

Mail Syntax when Mailing to Other Systems

The general form of syntax when mailing to a user on another system is either:

node!user

or

user@node

In the above example, *node* is the node name of the system the person is on, and *user* is the person's user name. Which syntax you use depends on how your system is configured to send mail. Ask your system administrator which syntax to use on your system.

Some Example Mail Addresses

Suppose you want to send mail to user **john,** who uses the system named **sell;** then you would use one of the following mail addresses (the exact syntax you'll use depends on how your system is configured to send mail):

In response to the `elm` prompt, enter

```
Send the message to: sell!john
```

or

```
Send the message to: john@sell
```

To send mail to **arnie** on your system, **john** on the **sell** system, and **leopold** on the **research** system, use one of the following addresses:

```
Send the message to: arnie  sell!john  research!leopold
```

or

```
Send the message to: arnie  john@sell  leopold@research
```

Deleting Mail Messages

After you've read your mail messages, you may want to delete them. You can mark messages for deletion using the `d` command. When you `quit elm` you have the option of deleting the marked messages.

Marking Messages to be Deleted

To delete one or more mail messages:

1. To delete a mail message, press `d` while the message is the current message. A `D` appears to the left of the message to show that it is marked for deletion.

   You can mark additional messages for deletion by making each message the current message and then pressing `D`. *Remember, to move to the next message, press* `j`, *to move to the previous message, press* `k`.

   The following screen shows two messages marked for deletion.

```
   Mailbox is '/usr/mail/leslie' with 4 messages. [Elm revision: X.X]


D  1   Apr 3   Leslie   (6)    Important Message to Myself
   2   Apr 3   Robert   (24)   Meeting Tommorrow
   3   Apr 2   Lynn     (154)  More Software Requests
D  4   Apr 2   Patrick  (78)   Hi there
```

2. When you enter `q` to `quit elm`, you'll be asked to confirm that you want any messages marked with a `D` deleted.

```
Delete messages? (y/n) y
```

   Press `y` or Return to delete the messages, or `n` to keep them. For example, press `y` or Return to delete messages 1 and 4 in the preceding example.

The next section tells you how you can store your messages in a different mailbox or save them to a file.

Saving a Mail Message to a File

When you `quit elm` you have the option of keeping your messages in the incoming mailbox (where your new messages arrive), or storing them in another mailbox (the default is *homedirectory*`/mbox`). While in `elm` you can also save messages to a designated file.

To save the current message to a file, at the `elm` command prompt, type:

```
Command: s
```

The following prompt appears;

```
Command: Save Message
File message in: =/username
```

If you press Return, the message is saved in a file named with your *username* in the `Mail` directory below your home directory. The default mail option is set so that the equal sign (=) is shorthand for *homedirectory*`/Mail`. If the `Mail` directory doesn't already exist, you need to create it.

If you want to save the message in another file, enter the name of the file. For example,

```
Command: Save Message
File message in: =/oldnews
```

Supposing that Leslie is the user, the current message is saved in the file `oldnews` in the `/user/leslie/Mail` directory. If the file already exists, the message will be appended to the contents of the file. If there is no existing `oldnews` file, one will be created.

After you save a message in a file, the message is marked with a `D` for deletion.

Viewing the Contents of Saved Messages

There are two ways to view the contents of messages saved in a file:

- The most common way to view saved messages is to change your mailbox and view them like you do in the incoming mailbox.

For example, assume that you have saved in the file **oldnews** two messages titled "More Software Requests" and "Meeting Tomorrow". To view these messages in the **oldnews** mailbox, use the **change** mailbox command. Enter:

```
Command: c
```

One or more prompts appear asking you whether you want to keep mail in your incoming mailbox or delete it. Answer **y** or **n** as you choose.

You're prompted next for the name of another mailbox.

```
Name of new mailbox: =/oldnews
```

*Enter the name of the file that contains the messages you want to see. Remember the = is shorthand for homedirectory/*Mail

The mailbox changes to **=/oldnews**. You can manipulate (read, edit, send, etc,) the messages in the **oldnews** mailbox just as you do messages in the incoming mailbox.

```
Mailbox is '=/oldnews' with 2 messages. [Elm revision: X.X]


1   Apr 3   Robert    (24)   Meeting Tommorrow
2   Apr 2   Lynn      (154)  More Software Requests
```

- A second way to view messages saved in a file is to use a shell command such as **more**. If you are in **elm** and want to access a shell command like **more** without leaving **elm**, use "!" to get to a shell prompt. For example, to examine the contents of the **oldnews** file, enter:

```
Command: !                                          Enter an exclamation
                                                    point (!)
Shell Command: more /user/leslie/Mail/oldnews       Enter a shell command.
```

Press Return to return to **elm**.

Customizing Elm

The `elm` mailer has different options you can set to make it more convenient for you to use. Among features you can change are the menus that appear on the screen, the printer your mail is sent to, and the order in which your mail is listed in your mailbox.

Bringing Up the Option Menu

To bring up the option menu, press `o` at the `elm` command prompt:

```
Command: o
```

You'll see a menu similar to the following:

```
C)alender file          : /user/leslie/calender
D)isplay mail using     : builtin
E)ditor                 : /usr/bin/vi
F)older directory       : /user/leslie/Mail
S)orting Criteria       : Date Mail Sent
O)utbound mail saved    : /user/leslie/mbox
P)rint mail using       : pr %s | lp
Y)our full name         : Leslie Pendergrast

A)rrow cursor           : OFF
M)enu display           : ON

U)ser level             : 0 (for beginning user)
N)ames only             : OFF

Select first letter of Option line, '>' to Save, or R)eturn

Command:
```

Figure 7-2. The Options Menu

This guide does not describe all of the options in the option menu. Rather, it gives you an example of changing an option; in this instance how to change the order in which your mail is listed.

Changing the Order of Your Mail Messages

To change the order in which your mail messages are listed in your mailbox, press `s` (the first letter of `S)orting criteria`) at the option menu prompt:

```
Command: s
```

You'll see a message indicating how messages are currently sorted. For example,

```
This sort will order most-recently-sent to least-recently-sent
```

To see different choices for sorting your messages, press the space bar. When you see the method you want press Return.

For instance, when you see:

```
This sort will order by sender name
```

Press Return, then press > to save the change.

The change is recorded in the `elmrc` file in the `.elm` directory below your home directory. This file controls many of the customized features of `elm`. (If the `.elm` directory does not exist, you need to create it.)

To return to your mailbox, press Return again.

The messages in your mailbox will now appear in alphabetical order by sender name.

To get information about a specific option in the option menu, type ? and then type the first letter of the option.

For More Information ...

For more information on the `options` command and the `elmrc` file see the *elm* entry in section 1 of the *HP-UX Reference*.

A Summary of Useful Mail Commands

While you are "in" `elm`, you can do many tasks, such as editing a mail message, replying to a message, or forwarding a message.

Table 7-1. Examples of Often-Used Mail Commands

The mail command	Does this ...
?	Get help on `elm` commands.
!	Allows you to send a command to the shell without leaving `elm`.
a	Set up mail aliases.
c	Change the mailbox.
d	Mark messages for deletion.
f	Forward the current message to another user.
g	Send a group reply to all recipients of the original message.
j	Move the message pointer to the next message.
k	Move the message pointer to the previous message.
m	Send mail to a specified user or users.
o	Allows you to alter the setting of different mail parameters, including the sorting method for messages, the destination of printed messages, the type of menus displayed, and so on.
p	Print messages. (You can change the destination of printed messages using the `o` command listed above.)
q	Quit `elm` with the option of changing the contents of the mailbox.
r	Reply to the author of the current message.
s	Save a message to a file.
x	Exit `elm` without making any changes.

To see a summary of all of the commands you can use from `elm`, type "?" at the `elm` command prompt.

You can abbreviate every mail command (except `help`) merely by specifying the first letter. For example, you can abbreviate the `delete` command with a single `d`. To abbreviate `help`, use a question mark (`?`).

For More Information ...

For more information on the `elm` commands see the following entries in section 1 of the *HP-UX Reference:*

- *elm*
- *readmail*
- *newmail*
- *elmalias*

8

Keeping Your System Secure

HP-UX provides many security features to protect files from unauthorized access. However, you need to follow good security practices to maintain security on your system. The degree to which you need to enforce security measures depends on where you work, your workplace's security policy, and the type of information with which you work.

Security Strategies

This chapter summarizes the security strategies you should follow to help keep your system secure:

- Become familiar with the security policies of your workplace.
- Keep your terminal secure.
- Choose a secure password, and protect your password after you've chosen it.
- Know who has permission to access your files and directories.
- Adequately protect sensitive files and directories from unauthorized access.

Securing Your Terminal

When you are working with sensitive material, take care to position your terminal so the screen is not visible to others. Never leave your terminal unattended: log off.

Guidelines for Securing Your Terminal

When working with sensitive material, take these security precautions:

- Position your terminal so the screen points away from open windows and doors.
- Never leave your terminal unattended:
  - Exercise care when logging in. Make sure no unauthorized person is observing you.
  - Log off if you will be away from your terminal for a long time (such as several hours or overnight).
  - Clear your display if you leave your terminal for a brief period. Type `clear` at the command line prompt. (The `clear` command clears only the current screen; you can still scroll up and see information.)

Note Check the security policies of your workplace. You may be required to log off whenever you leave your terminal, even if only for a brief period.

Working in an Audited Environment

HP-UX provides the capability to audit computer use, both on an individual and system-wide basis. Depending on how your system is configured, your actions may be recorded by an audit program. This program monitors user actions at your terminal and records security-relevant information.

Choosing a Secure Password

When you choose a password, you want to ensure that no one can guess what you chose. If someone knows your password, that person may log in and access your files. This section offers suggestions on how to select and protect your password. These guidelines are of particular significance if you work with sensitive material.

Selecting a Secure Password

When selecting a password in a secure environment, follow these guidelines:

- Choose a word that isn't publicly associated with you (your personal or professional life, your hobbies, etc.):
  - Don't use your name, your spouse's name, your children's names, or your pets' names.
  - Don't use the name of your street or your car.
  - Don't use phone numbers or special dates (anniversaries, birthdays, etc.).
  - Don't use your address, social security number, or license plate numbers.
- Choose a word that isn't listed in the dictionary (spelled either forwards or backwards). Password-cracking programs can use dictionary lists.

What *can* you use as a password? Here are a few suggestions:

- Make up a nonsense word.
- Make up an acronym.
- Misspell a word intentionally.
- String together syllables from a favorite song or poem.

Note HP-UX requires that your password be six to eight characters long. At least two of these characters must be letters (uppercase or lowercase); at least one character must be either a numeral (the digits 0 through 9) or a special character (such as -, _, or $).

Protecting Your Password

When you have chosen your password, follow these guidelines to ensure that no one discovers it:

- Never write down your password.
- Don't tell others your password.
- Don't let others watch as you type your password.
- Don't store your password in the function keys of a terminal.
- Change your password occasionally (for example, once every three or four months). Refer to "Setting Your Password" in Chapter 2 if you need information on how to change your password.
- If you use more than one computer, use a different password for each system.

Protecting Your Files and Directories

> **Access permissions** determine who can access your files and directories and the type of access allowed. You should always be aware of the permissions assigned. Check your files and directory permissions periodically to make sure appropriate permissions are assigned. If you find any unfamiliar files in your directories, report them to the system administrator or security officer.

Always carefully consider the permissions you allow on your files and directories. Give others access to them *only* when you have good reason to do so (if you are working on a group project, for example, your group may need access to certain files or directories.)

As you learned in Chapters 3 and 4, the basic access permissions assigned to files and directories distinguish between three classes of users: owner, group, and other.

Each of the classes of users can access files or directories in any of three ways: read, write, and execute/search (`r`, `w`, and `x`).

Access to Sensitive Files

Make sure that permissions assigned to sensitive files and directories are appropriate. Here are some general suggestions:

- Only you should be able to write to your home directory.
- Only you should be able to write to the files used to customize your home environment, for example, `.login` and `.profile`. (These files are discussed in the section "Invisible File Names" in Chapter 3.)
- Only you should be able to write to your mailfile `/usr/mail/`*username*.

Listing Permissions with the ll Command

In Chapters 3 and 4, you learned to use the `ll` command to see the basic access permissions assigned to files and directories. This section contains a brief summary of how the `ll` command works.

To see the access permissions, owner name, and group name on `myfile`, type the following:

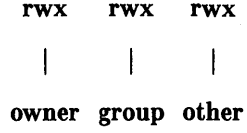
```
$ ll myfile
```

When you press (Return), you should see something like this:

```
-rw-r--r--  1  leslie  users      154 Nov  4 10:18 myfile
```

The first dash in the long listing indicates that `myfile` is a file. (If `myfile` were a directory, you would see a `d` in place of the dash.) The next nine positions indicate read, write, and execute permissions for *owner, group,* and *other.* If a permission is not allowed, a dash appears in place of the letter.

Here is a closer view with all permissions indicated (note that the permissions are in "sets" of three):



For More Information . . .

For more detail on the `ll` command, see the section in Chapter 3 "Finding Out Who Can Use Your Files" or the section in Chapter 4 "Finding Out Who Can Use Your Directories".

Changing Who Has Access to Files

If you want to change the basic access permissions assigned to a file use the `chmod` (**ch**ange **mod**e) command to control who has read, write and execute permission to your files.

You can change file permissions using the `chmod` command. The `chmod` command sets a file's read, write, and execute permission for you, the file's group, and other users. Before using this command, you should always carefully consider what file permissions you give to others.

In general, give others access to your files *only* when you have good reason to do so (if you are working in a group project, for example, your group may need access to certain files).

Using chmod to Set File Permissions

To illustrate the `chmod` command, you can set `myfile`'s permissions so that *only* you can read from and write to the file. The general syntax of `chmod` is as follows:

`chmod` *number file_name*

Number is a three-digit number specifying how you want to protect the file. The three digits sequentially set permissions for each of the three groups: the owner, the group to which the owner belongs, and all other users. The *file_name* is the name of the file you want to protect.

For example, to set the access permissions on `myfile` so that you, your group, and other users can read myfile, but no one can write to it, enter:

```
$ chmod 444 myfile
```

The numerical notation that `chmod` uses to set permissions is based on octal values, as follows:

```
r  w  x

1  1  1  =  permissions allowed (rwx is output)

0  0  0  =  permissions denied  (--- is output)
```

The following diagram shows the various combinations of permissions and the corresponding octal values. These values are used by `chmod` to change file and directory permissions.

```
---   --x   r--   r-x   rw-   rwx
000   001   100   101   110   111
 0     1     4     5     6     7
```

The next section shows you some examples of frequently used `chmod` commands.

Common Ways to Use chmod to Protect Files

The preceding section introduced the **chmod** command and explained the syntax used to protect files. This section gives you some examples of common ways to use **chmod**.

Table 8-1 shows various **chmod** commands you can use to protect **myfile**.

Table 8-1. Common Uses of chmod

To protect myfile so that ...	Use ...
Only you can read from **myfile**, and no one (including you) can write to it. Set permissions to **-r--------**.	**$ chmod 400 myfile**
Everyone can read from **myfile**, but no one can write to it. Set permissions to **-r--r--r--**.	**$ chmod 444 myfile**
Only you can write to **myfile**, but everyone can read it. Set permissions to **-rw-r--r--**.	**$ chmod 644 myfile**
Only you and members of your group can read and write to **myfile**, but everyone can read it. Set permissions to **-rw-rw-r--**.	**$ chmod 664 myfile**
Everyone can read from or write to **myfile**. Set permissions to **-rw-rw-rw-**.	**$ chmod 666 myfile**
Only you can read from or write to **myfile**, but no one else can. Set permissions to **-rw-------**	**$ chmod 600 myfile**

For example, suppose you want to protect **myfile** so that neither you nor anyone else can modify it, but everyone can still read from it. Then use **chmod** as shown in the second entry in Table 8-1:

```
$ chmod 444 myfile
```

Run the **ll** command to verify that **myfile** has read permission only:

```
$ ll myfile
-r--r--r--  1  leslie  users      154 Nov  4 10:18 myfile
```

With only read permission on **myfile**, no one can write to it. Also, if you now try to remove **myfile**, the **rm** command asks you whether you really want to remove the file:

```
$ rm myfile
myfile: 444 mode? (y/n) n
```

You do not want to remove it, so enter **n**. *(If you did want to remove it, you would enter* **y**.*)*

Later, if you want to permit yourself and other members of your group to read from and write to **myfile**, use **chmod** as follows:

```
$ chmod 664 myfile
```

The **ll** command now should show:

```
$ ll
-rw-rw-r--  1  leslie  users      154 Nov  4 10:18 myfile
```

For More Information ...

This section covered some of the most common uses of the **chmod** command for protecting files. To learn more about **chmod**, refer to the following books:

- The system administrator's manual for your system.
- The *chmod* entry in section 1 of the *HP-UX Reference*.

Changing Who Has Access to Directories

In addition to changing permissions on files, the `chmod` command can also change permissions on directories. Using `chmod`, you can control who has access to your directories, and what kind of access they have. For example, you can protect a directory so that no one can list its files. Or you can control whether users can remove or change files in a particular directory.

Table 8-2 defines some of the more common uses of `chmod` with directories. All the examples in this table assume that the directory `projects` exists under your current working directory.

Table 8-2. Setting Directory Protection for the projects Directory

To Set Permissions to ...	Use this command
Allow other users to list and access the files in `projects`, but not to create or remove files from it. Set permissions to `drwxr-xr-x`.	`$ chmod 755 projects`
Allow all users to list, create, remove, and access files in `projects`. Set permissions to `drwxrwxrwx`.	`$ chmod 777 projects`
Allow only yourself to list, create, remove, and access files in `projects`. Set permissions to `drwx------`.	`$ chmod 700 projects`

Note When determining who should be allowed to use your directories, be aware that *anyone who can write to a directory also can remove or rename a file in that directory*—even if that person cannot write to the file.

Finding Out Default Access Permissions

In the preceding section you learned how to change the permissions on individual files and directories using the `chmod` command. You should also be aware of the default permissions assigned to all of your files and directories at the time you create them. You can list or change these default settings by using the `umask` command.

Default file permissions are assigned by the system when you create a new file or directory. To find out what these permissions are, create a new file or directory and then use the `ll` command to examine the permissions assigned.

You may want to reset the default permissions for your home directory to make access to your files and directories either more or less restrictive. You can change how the default file permissions are set using the `umask` command.

The `umask` command works in an opposite manner to the `chmod` command. Suppose the system default assigned to a new directory is 777 (read, write, and execute/search). The three digits you specify with the `umask` command are subtracted from that number. For example, suppose you enter:

```
$ umask 022
```

The new permissions are set to 755. Each of the digits specified with `umask` (022) is subtracted from the corresponding default access permissions (777) to get 755. Therefore, the next time you create a new directory, it is assigned the permissions 755 (`drwxr-xr-x`).

To find out what `umask` has been set on your system, type

```
$ umask
```

Here are some examples of common settings for `umask`:

`umask 022` Assigns permissions so that only you can read or write to files or directories that you own. All others have read permission only to your files.

`umask 002` Assigns permissions so that you and also members of your group have read and write permission to files and directories that you own. All others have read permission only.

If you set the `umask` command at a prompt, it will apply for the current login session only. To apply a `umask` to all subsequent files and directories you create, add the `umask` command to your `.login` file (C-shell users) or `.profile` file (Korn and Bourne shell users). For more information on these files, see *A Beginner's Guide to Using Shells.*

For More Information ...

To learn more about the `umask` command refer to the *umask* entry in section 1 of the *HP-UX Reference*).

To learn more about the `.profile` and `.login` files see *A Beginner's Guide to Using Shells.*

To learn more about the `ll` command, refer to the *HP-UX Reference* (you'll find the `ll` command listed under *ls* in section 1).

9

Other Common HP-UX Tasks

The previous chapters of this guide tell you the basic things you need to know about HP-UX. In addition, there are many other useful HP-UX commands and utilities that you may want to use. This chapter tells you about some of these useful commands.

Specifically this chapter discusses:

- Saving files on tape using `cpio` and `tcio`.
- Retrieving files from tape using `cpio` and `tcio`.
- Getting print job information or canceling a print job.
- Searching for files using `find`.
- Searching for text patterns using `grep`.
- Running commmands automatically at preset times using `at` and `crontab`.
- Sorting files using `sort`.

For more information on any of commands discussed in this chapter see the appropriate *HP-UX Reference* entry or the corresponding on-line man-page.

Saving Files on Tape Using cpio and tcio

With the `cpio` and `tcio` commands you can save directories and files to tape. Use these commands when you want to exchange files with other HP-UX systems via tape or keep your own tape archives, distinct from system backups.

Overview

Typically, your system administrator will assume responsibility for making full system backups. He or she will regularly copy files and directories on the system to tape or another disk, as protection against loss of data. However, you may also want to place files and directories on tape, either to transfer them to another system or to make a personal backup tape.

You can use `cpio` to save files to both cartridge and 9-track magnetic tape. Whenever you use `cpio` to save files to cartridge tape, you should use it with `tcio` to improve performance and reduce wear on the tape drive.

Note The `cpio` command will *not* preserve optional ACL permissions if they have been assigned to files. For more information on ACLs see Appendix A.

Here is an overview of the commands used to save files to tape.

`find . -print`	Generates a list of path names of all files and directories under the current directory; sends the list to `cpio`.
`cpio -ocv`	Receives a list of path names from the `find` command; then generates ASCII-format file information.
`tcio -o`	Receives file archive information from `cpio` and stores it on tape. The `tcio` command is used only with cartridge tape.

Creating a Tape Archive

The following steps tell you how to save all files and directories under the current directory to tape.

Caution Using `cpio` to save files to tape overwrites the tape's previous contents.

---

1. Get permission from your system administrator to use the tape drive, and ask for the tape's device file name.
2. Mount a tape in the tape drive. If you don't know how to mount a tape, ask your system administrator or see the manual that comes with the tape drive.
3. `cd` to the directory containing the files and subdirectories you want saved on tape.
4. Run one of two commands following, depending on whether you are copying to cartridge or 9-track magnetic tape. Use the device file name obtained in step 1:

   If you are using a cartridge tape, enter:

   ```
   $ find . -print | cpio -ocv | tcio -o device_file
   ```

   If you are using a 9-track tape, enter:

   ```
   $ find . -print | cpio -ocv > device_file
   ```

Example of Creating a Tape Archive

Assuming that the tape device file for your system is `/dev/update.src`, here is how you would save all the files and subdirectories contained in your home directory to cartridge tape:

```
$ cd
cd to your home directory
$ ls *
list its contents to verify that it's O.K.
memos:
report1  c2.beg.gd  titles  report2  design1

projectX:
plan mail stuff

$ find . -print | cpio -ocv | tcio -o /dev/update.src
The device file name may vary depending on your system.
```

If you are copying to 9-track tape, enter:

```
$ find . -print | cpio -ocv > /dev/rmt/0m
The device file name may vary depending on your system.
```

Selectively Saving Files to Tape

You can use the `ls` command with `cpio` to save only specified files to tape.

Caution Remember that whenever you write files to tape with `cpio` you overwrite the tape's previous contents.

For example, to save files `report1` and `report2` in the current directory, enter the following commands:

To save to cartridge tape, enter:

```
$ ls report1 report2 | cpio -ocv | tcio -o device_file
```

To save to 9-track tape, enter:

```
$ ls report1 report2 | cpio -ocv > device_file
```

Listing Files on Tape

To examine the files listed on tape, use the `t` option.

To list files on cartridge tape, enter:

```
$ tcio -i device_file | cpio -ictv
```

To list files on 9-track tape, enter:

```
$ cpio -ictv < device_file
```

Retrieving Files from Tape Using cpio and tcio

Use the `cpio` and `tcio` commands to retrieve directories and files previously saved to tape and copy them to your system disk.

Overview

Here is an overview of the commands used:

`tcio -i`	Generates a list of path names of all files and directories on the tape; sends the list to `cpio`. The `tcio` command is only used with cartridge tape.
`cpio -icdv`	`cpio` receives archive information either from `tcio` or directly from the tape and copies files to the system disk. The archive must have been previously created using `cpio` with the `-o` option.

Retrieving All Files from Tape

1. Get permission from your system administrator to use the tape drive, and ask for the tape's device file name.
2. Mount a tape in the tape drive. If you don't know how to mount a tape, ask your system administrator or see the manual that comes with the tape drive.
3. `cd` to the directory where you want to copy the files and subdirectories previously saved on tape.
4. Run either of the following commands depending on whether you are using a cartridge tape or a 9-track magnetic tape. Use the device file name obtained in step 1:

   If you are using a cartridge tape, enter:

   ```
   $ tcio -i device_file | cpio -icdv
   ```

   If you are using a 9-track tape, enter:

   ```
   $ cpio -icdv < device_file
   ```

Note When you are retrieving files from tape, the `cpio` command will not overwrite files on your disk that have been modified more recently than the files on tape. If you want to override this feature and copy over newer files, use `cpio` with the `-u` option.

Example of Retrieving Files from Tape

Assuming that the tape device file for your system is `/dev/update.src`, here is how you would copy files and subdirectories contained on cartridge tape to your home directory:

```
$ cd                                          cd to your home directory
$ tcio -i /dev/update.src | cpio -icdv        The device file name may vary
                                              depending on your system.
```

To copy files from 9-track tape, enter:

```
$ cd                              cd to your home directory
$ cpio -icdv < /dev/rmt/0m        The device file name may vary
                                  depending on your system.
```

Note The directories you are restoring from tape must already exist on your system. If they don't, use `cpio` with the `-d` option (as shown in the previous examples) and it will create them.

Selectively Retrieving Files from Tape

If you wish, you can retrieve only selected files. For example, to retrieve files that match the pattern `report*`, use the following commands:

If you using a cartridge tape, enter:

```
$ tcio -i device_file | cpio -icv 'report*'
```

If you using a 9-track tape, enter:

```
$ cpio -icv 'report*' < device_file
```

You must enclose the text pattern '`report*`' in single quotes.

Table 9-1.
Summary of Common cpio Commands for Cartridge Tape

To do this ...	Use this command ...
Save files in the current directory and its subdirectories to cartridge tape.	`find . -print | cpio -ocv | tcio -o` *device_file*
Selectively save files to cartridge tape.	`ls` *file1 file2* `| cpio -ocv | tcio -o` *device_file*
List files on cartridge tape.	`tcio -i` *device_file* `| cpio -ictv`
Retrieve files from cartridge tape.	`tcio -i` *device_file* `| cpio -icdv`
Selectively retrieve files from cartridge tape.	`tcio -i` *device_file* `| cpio -icv` *file1 file2*

Table 9-2.
Summary of Common cpio Commands for 9-track Magnetic Tape

To do this ...	Use this command ...
Save files in the current directory and its subdirectories to 9-track tape.	`find . -print | cpio -ocv >` *device_file*
Selectively save files to 9-track tape.	`ls` *file1 file2* `| cpio -ocv >` *device_file*
List files on 9-track tape.	`cpio -ictv <` *device_file*
Retrieve files from 9-track tape.	`cpio -icdv <` *device_file*
Selectively retrieve files from 9-track tape.	`cpio -icv` *file1 file2* `<` *device_file*

Getting Information About Print Jobs

After you send a file to the printer, you may want to find out the status of your print request or cancel it. This section tells you how to manage your print jobs.

Overview

Many people can send a printing job to the printer at one time. To make sure that the print requests don't interfere with each other, the print **spooler** handles the requests and makes sure they print one at a time. Each print request is given a identification number. The jobs are printed one at a time, in the order in which they are requested.

Sending Files to the Printer

You can print a file using the `lp` (line print) command.

```
$ lp file_name
request id is pr1-5453 (1 file)
$
```

The `lp` command sends the *file* to the spooler, which stores the file until its turn to be printed. The request id consists of two parts: the printer name and the print request number. In this example, the name of the default printer is `pr1`; the print request number is 6.

The printing process takes place in the background, so after you enter the `lp` command, the shell prompt returns and you can enter another command.

You can send more than one file to the printer at a time. For example, suppose you want to send multiple files to the printer. Assuming the files are called `report1`, `report2`, and `report3`, enter:

```
$ lp report1 report2 report3
```

You can also use a filename pattern and enter:

```
$ lp report*
```

The files are sent sequentially to the printer.

Sending Files to an Alternate Printer

Your system may be connected to more than one printer. If you want to print to a printer other than the default printer, use `lp` with the `-d` option. For example, enter:

```
$ lp -dpr2 file_name
```

This example sends *file_name* to the printer called `pr2`. To find out the names of the printers connected to your system, ask your system administrator.

Finding Out the Status of Your Print Request

After you make a printing request, you may want to find out its status. Use the `lpstat` command to get information about the spooler. For example, enter:

```
$ lpstat
market: ready and waiting
pr1-5453  leslie   2034   Jul 18 17:26
```

The example shows that job #5453 is waiting to be printed from the `market` system on printer `pr1`.

To find out the position of your job in the printer queue use `lpstat` with the `-t` option. Enter:

```
$ lpstat -t
```

You'll see a display showing you information about the available printers on the system. At the end of the display, the jobs in the queue are listed for the different printers.

```
$ lpstat -t
scheduler is running
system default destination is pr1
device for pr1 is /dev/null
device for pr2 is /dev/tty0p4
pr1 accepting requests since April 10 2:14
pr2 accepting requests since July 12  3:45
printer pr1 is now printing
printer pr2 is now printing

printer queue for pr1: ready and waiting
pr1-5450  scott    14379  Jul 18  17:15  on pr1
pr1-5451  joanne   16786  Jul 18  17:21  on pr1
pr1-5453  leslie   2034   Jul 18  17:26  on pr1

printer queue for pr2: ready and waiting
pr1-5452  michael  1098546  Jul 18  16:59  on pr2
```

Notice that Leslie's job is scheduled on `pr1` behind jobs for Joanne and Scott.

Canceling a Print Job

To cancel a print job, enter:

```
$ cancel request_id
```

where *request_id* is the job number of your print request.

For example, if Leslie wanted to cancel her print request, she would enter:

```
$ cancel pr1-5453
```

Searching for Files using find

You can use the `find` command to search through a directory and its subdirectories for files meeting certain criteria. You can then execute a command on the files you've found. To display the output of `find` on the screen, you must use the `-print` option.

Finding Files that Match a Pattern

Suppose you want to display all files in the current directory and its subdirectories that begin with `d`. Enter:

```
$ find . -name 'd*' -print
```

The dot (.) causes `find` to search the current directory and its subdirectories. The `-name` option followed by a filename or a filename pattern (in this case `d*`) tells `find` to search for all filenames that begin with `d`.

Note that `d*` is enclosed by single quotes `'d*'`. If you use a file name pattern in the `find` command, you must quote it so that the shell will interpret it correctly. The `-print` option displays the output on the screen.

Finding Files Newer than a Certain File

Suppose you want to display all files modified after a certain file. To display all files newer than `myfile` in the `/user/leslie` directory and its subdirectories, enter:

```
$ find /user/leslie -newer myfile -print
```

This example can be read as follows: find in directory `/user/leslie` and its subdirectories all files modified after `myfile` and display the output on the screen. (To find out the date and time a file was last modified use the `ll` command.)

Running Commands on Files

You can execute commands on files located with the `find` command. Let's say you want to remove all files with a `.tmp` extension in the current directory and its subdirectories. Enter:

```
$ find . -name '*.tmp' -print -exec rm {} \;
```

This example finds and displays on the screen all files in the current directory and its subdirectories that end in `.tmp`, then deletes these files. The `-exec` option causes the following command (`rm`) to be executed. The brackets { } represent the files found with the `find` command. The semi-colon that ends the command string is escaped with a backslash (\;).

Searching for Text Patterns Using grep

You can use the grep command to search for a text pattern within a file or to display the names of files that contain a specified text pattern. This command is useful when you want to search for information in files or directories.

Overview

The grep command looks at each line of one or more files for a text string that matches a specified pattern. When it finds a matching text string, it displays the line in which the matching string is found.

Searching a File for a Text String

Suppose you have a mailing list called `mailist` with the contents shown below:

```
Smith, Joe        2345 Pine St.      Santa Clara, CA
Walsen, Stacy     493  Winkle Ave.   San Jose, CA
Diaz, Robert      6789 Pine St.      Santa Clara, CA
Wang, Michael     1832  Jackson St.  Santa Clara, CA
```

If you want to extract the addresses of all the people on `Pine St`. Enter:

```
$ grep Pine mailist
```

The grep command lists all lines in `mailist` that contain the string `Pine`. The output is:

```
Smith, Joe        2345 Pine St.      Santa Clara, CA
Diaz, Robert      6789 Pine St.      Santa Clara, CA
```

Searching Multiple Files

The `grep` command can be useful in other ways. Sometimes, you want to find information, but you don't know or can't remember in which file it's located.

Suppose you have three mailing lists, and can't remember which contains Stacey Walsen's address, enter:

```
$ grep 'Walsen, Stacey' mailist mailist2 mailist3
mailist: Walsen, Stacy    493  Winkle Ave.  San Jose, CA
```

The `grep` command displays the line containing Stacey's address and the file in which it was found. Note that because it contains a space, the string must be surrounded by single quotes (`'Stacey, Walsen'`). If `grep` had found other instances of the specified text string, it would list each instance that it found.

Running Commands at Preset Times using at and crontab

The `at` and `crontab` commands are useful if you want to run resource-intensive commands when demands on the system are low or routinely run commands at certain times. For example, you can schedule a long file to print at midnight, or arrange that temporary files in your home directory be erased every day.

Overview

Here is an overview of the commands.

`at`	runs commands in your home directory at the time you specify.
`crontab`	runs commands in your home directory at regularly specified intervals.

Prerequisites

Before you can use `crontab` or `at`, your system administrator must set up certain files that allow you permission to run these commands. These files are located in the directory `/usr/lib/cron`.

Two files called `at.allow` and `at.deny` in `/usr/lib/cron` determine whether you can use the `at` command. You are permitted to use `at` if your username appears in the file `at.allow`. If that file does not exist, the system checks `at.deny` to determine if you are listed there and should be denied access to `at`.

If neither file exists, only someone with system administrator privileges is allowed to use `at`. If only `at.deny` exists and is empty, all users are allowed to use `at`.

Permission to use the `crontab` command is determined the same way, except the files are called `cron.allow` and `cron.deny`.

For more information see the entries *at* and *crontab* in section 1 of the *HP-UX Reference*.

Running Commands at a Specified Time using at

Suppose you have a large file that you want to print, but you don't want to tie up system resources during the day to do it. Use the `at` command to print the file at 4:00 AM.

```
$ at 4am          Enter the at command
lp big_file       Enter the command you want to execute
CTRL-D            To end the command, press CTRL-D
```

You can specify a date as well as a time. For example, if you want to print a file on January 10th at 3:30AM, enter:

```
$ at 3:30am Jan 10
lp bigfile
CTRL-D
```

Note Any output or error messages generated from using the `at` command are sent to you in a mail message, unless you redirect them to a file.

Listing the at Command

To list a job scheduled with `at`, enter:

```
$ at -l
job 617 at wed jan 10 03:30:00 1990
```

The number of the job and the time it's scheduled to be executed are displayed.

Removing the at Command

To remove a job scheduled with `at` use the `-r` option and specify the job number. For example, to cancel the job shown in the preceding example, enter:

```
$ at -r 617
```

Running Commands at Regular Intervals using crontab

You can use the `crontab` command to run commands at regular intervals. For example, you might want a reminder about a weekly meeting to appear in your mailbox, or to erase all files with a `.tmp` extension from your home directory every day.

The `crontab` command creates a file called by your username in the directory `/usr/spool/cron/crontabs`. (The *username* is the name you use to login.) The commands in the file are executed at the specified intervals. The `crontab` command file runs in your home directory. Any output or error messages from executed commands are sent to you in a mail message, unless you redirect them to a file.

A `crontab` file contains lines divided into six fields each. The fields are separated by spaces or tabs. The first five fields represent the time the command will be run.

minute (0-59)
hour (0-23)
date of the month (1-31)
month of the year (1-12)
day of the week (0-6 with 0=Sunday)

The sixth field is a string that is executed at the appropriate time.

Using crontab to Create a Command File

To create a `crontab` command file (overwriting any previously created file), enter:

```
$ crontab
```

Type the commands you want and then press CTRL-D. For example,

```
30  8  * * 4  echo "Staff meeting today at 10:00AM"
0   0  * * *  rm *.tmp 2> errfile
CTRL-D
```

The `crontab` file is interpreted as follows:

- Line 1 - On Thursday at 8:30AM `crontab` sends you a reminder of your 10:00AM staff meeting. The first field (30) indicates 30 minutes past the hour. The second field (8) indicates the hour. The third and fourth fields contain asterisks (*), which indicate all legal values. The fifth field (4) indicates Thursday.

  The `crontab` command sends output and error messages to your mailbox (unless you redirect them to a file). Every Thursday morning at 8:30AM you will receive a mail message:

```
Staff meeting today at 10:00AM

*********************************************
Cron: The previous message is the standard output
and standard error of one of your cron commands.
```

- Line 2 - At midnight every day `crontab` erases files with a `*.tmp` extension in your home directory. Any error messages are redirected to `errfile` in your home directory. *Note:* If the `crontab` file runs in a C Shell you redirect output and error messages using `>& outfile`.

Listing the crontab File

You can get a listing of the contents of your `crontab` file by entering:

```
$ crontab -l
30  8  * * 4  echo "Staff meeting today at 10:00AM"
0   0  * * *  rm *.tmp 2> errfile
```

Removing a crontab File

You can remove your crontab file by entering:

```
$ crontab -r
```

Your `crontab` file is removed from the directory `/usr/spool/cron/crontabs`.

Using sort to order files

You can use the `sort` command to order the contents of files. You can sort alphabetically, numerically, or by different fields.

Overview

The `sort` command sorts a file on a line-by-line basis. It compares the first characters in each line; if they are the same, it compares the next two characters, and so on through the end of each line.

The `sort` command also recognizes different sortable fields. Fields are separated from each other by spaces or tabs. If there is more than one leading space before a field, `sort` will count each space as a sortable character.

Sort Files in Alphabetical Order

Here is an example file called `list`, containing the names and telephones numbers of the members of a tennis club and the number of tickets they have sold for a benefit match. Each line in the file has four fields: the first name, the last name, the number of tickets sold, and the phone number. The fields are separated by spaces.

If you want to practice using the `sort` command, type in the `list` file using the `cat` command as shown following:

```
$ cat > list
Nancy Smith      4    245-1342
Jeff Bettleman   8    438-7689
Jeff Plimpton    13   729-8965
Joyce Smith      6    467-2345
```

To `sort` the file, enter:

```
$ sort list
Jeff Bettleman   8    438-7689
Jeff Plimpton    13   729-8965
Joyce Smith      6    245-1342
Nancy Smith      4    467-2345
```

Note that the list is sorted alphabetically by *first* names. The `sort` command without any options will sort starting with the first field in each line (in this example the first name).

The `sort` command places the two "Jeffs" in the correct order, because after comparing the first names and finding them identical, it compares the second fields (last names).

Sorting Files by Different Fields

The next example shows how you can use `sort` to order the list by last names.

```
$ sort +1 list
Jeff Bettleman  8    438-7689
Jeff Plimpton   13   729-8965
Nancy Smith     4    467-2345
Joyce Smith     6    245-1342
```

The `+1` option causes `sort` to skip the first field of each line and start sorting at the second field (the last names). However, note that the first names Nancy and Joyce are not in alphabetical order.

The next example shows how to sort by last name and then by first name.

```
$ sort +1 -2 +0 list
Jeff Bettleman  8    438-7689
Jeff Plimpton   13   729-8965
Joyce Smith     6    245-1342
Nancy Smith     4    467-2345
```

The `+1` causes `sort` to skip the first field (first name) and sort by the second field (last name). The `-2` causes the `sort` command to stop after comparing the second fields. If any of the second fields are identical (in this case the Smiths), the `+0` instructs `sort` to make a second pass and sort from the beginning of the line (putting the first names in correct order).

Note that the `sort` command will only make a second pass at comparing lines if its first pass shows all of the examined fields in the lines to be identical. If `-2` were not specified in the above example, `sort` would show the third fields (telephone numbers) to differ and would never make a second pass to compare the first names.

Sorting in Numerical Order

Suppose you want to sort the file by the number of tickets sold. Enter:

```
$ sort -rnb +2 -3 list
Jeff Plimpton     13  729-8965
Jeff Bettleman    8   438-7689
Joyce Smith       6   245-1342
Nancy Smith       4   467-2345
```

There are a couple of things to keep in mind when sorting in numerical order. First, **sort** will consider multiple leading blanks in its comparison. Since there are a variable numbers of blanks preceding the third field (ticket number), your results will be incorrect unless you use the **-b** (ignore leading blanks) option.

Also, the numbers will not be in the right order unless you use the **-n** (numeric) and **-r** (reverse) options. The **sort** command by default sorts from least to greatest numbers so you need the **-r** option to list the greatest number first. Also, by default **sort** works by comparing characters in a field in the order they appear. So without the **-n** option, 8 would come before 13, because 8 is larger than 1.

Table 9-3. Summary of Common sort Options

This option	Does this ...
`-b`	(disregard leading blanks) Blanks, which can be either spaces or tabs, delimit sortable fields. Without the `-b` option, `sort` considers all blanks preceding a field to be part of that field, and will use them in sort comparisons.
`-d`	(dictionary) disregards all characters that are not alphanumeric or blanks. In particular, this option disregards control characters or punctuation.
`-f`	(fold) doesn't consider any difference between upper and lower case.
`-n`	(numerical) sorts in numerical order. Plus and minus signs are considered as plus and minus signs, and dots (.) are considered as decimal points.
`-r`	(reverse) reverses sort order (i.e., z-a).
`-u`	(unique) removes duplicate lines from a sorted file.

A

Access Control Lists (ACLs)

If you are working in an environment where security is especially important, you may want to restrict access to files selectively. You can specifically control who has access to files and directories by assigning optional **access control list (ACL)** permissions.

Caution Do not use ACLs until you have checked with your system administrator.

For the purpose of selectively restricting access to files with ACLs, users can be classified into:

- **individual users**—any individual user on the system
- **specific groups**—groups other than the owner's group.

For example, you may want to assign ACL permissions to restrict access to a sensitive file so that only you and your manager can read it. You also may want to restrict access to a sensitive directory so that only certain members of a group can write to it.

Listing Permissions for Specific Users and Groups.

In the previous sections of this guide, you learned how to use the `ll` and `chmod` commands to control which classes of users have access to your files and directories. Optional **Access control lists** (ACLs) let you control access rights for specific lists of users or groups.

Using the ll command

The `ll` command indicates whether ACLs have been assigned to a file or directory. If you see a plus sign (+) at the end of the listing (for example, `-rw-r--r--+`), it means that access control list (ACL) entries have been assigned. To find out specific information on ACLs, use the `lsacl` command.

Using the lsacl Command

To see exactly who can access certain files and directories and what permissions are allowed, use the `lsacl` command. The general form of the `lsacl` command is as follows:

```
$ lsacl file_name
```

The system will respond with a listing in this general form:

(user.group,mode) . . . file_name

where *(user.group,mode)* is an ACL entry (there will always be at least three ACL entries), and *file_name* is the name of the file or directory for which you want a listing. The following table describes what each element of an ACL entry means:

This element ...	Means ...
user	The user's login name; a percent sign (%) in this position means all users.
group	The user's group; a percent sign (%) in this position means all groups.
mode	The permissions allowed: read (r), write (w), execute/search (x). A dash (-) means a permission is denied (for example, rw- means that read and write permissions are allowed, but execute/search permission is denied).

An Example Using the lsacl Command

Suppose you run the `lsacl` command on `myfile`:

```
$ lsacl myfile
(sally.adm,rw-) (leslie.%,r--) (%.mtg,r--) (%.%,---) myfile
```

Interpret the above listing as follows:

`(sally.adm,rw-)`	The user `sally` while in the group `adm` has read and write permissions (`rw-`) on `myfile`.
`(leslie.%,r--)`	The user `leslie` while in any group (`%`) has read permission (`r--`) on `myfile`.
`(%.mtg,r--)`	Any user (`%`) in the group `mtg` has read permission (`r--`) on `myfile`.
`(%.%,---)`	No other user (`%`) from any other group (`%`) has read, write, or execute permissions (`---`) on `myfile`.

The following section explains how you can change the permissions on ACLs.

Changing Permissions for Specific Users and Groups

In the previous sections of this guide, you learned how to use the `chmod` command to change general permissions on your files and directories. By adding entries to optional access control lists (ACLs), you can allow or deny file and directory access to particular users or groups of users. Set and change ACLs with the `chacl` (**ch**ange **acl**) command.

Before using the `chacl` command, always carefully consider the permissions you allow on your files and directories. Give others access to your files *only* when you have good reason to do so (if you are working on a group project, for example, your group may need access to certain files).

How the chmod and chacl Commands Interact

Since you can use both the `chmod` and the `chacl` commands to change access permissions, you need to be aware of how the two commands interact.

- The `chacl` command is a superset of the `chmod` command. Any specific permissions you assign with the `chacl` command are added to the more general permissions assigned with the `chmod` command.

  For example, suppose you use the `chmod` command to allow only yourself write permission to `myfile`. You can use the `chacl` command to also allow your manager write permission to `myfile`. Users other than yourself and your manager will still be denied write permission as previously specified by the `chmod` command.

- If you use `chmod` after assigning additional permissions with `chacl`, the additional permissions are deleted unless you use the -A option of `chmod`. For example, to keep any existing ACL permissions on the file `myfile`, enter

  ```
  chmod -A number myfile
  ```

Using the chacl Command

The general form for the `chacl` command is as follows:

```
$ chacl 'user.group operator mode' file_name
```

user — Indicates the user's login name; a percent sign (%) in this position means all users.

group — Indicates the user's group; a percent sign (%) in this position means all groups.

operator — Indicates whether you are adding or denying permissions to existing ACL entries or whether you are adding new ACL entries. A plus sign (+) adds permissions; a minus sign (−) denies permissions; and equals sign (=) means "this permission exactly," and usually is used when adding new ACL entries.

mode — Indicates the permissions allowed. Possible modes are read (`r`), write (`w`), and execute/search (`x`). An *operator* (explained above) immediately precedes the mode (for example, `+rw` adds read and write permissions; `−rw` denies read and write permissions).

file_name — Indicates the name of the file or directory for which you want to specify access.

Examples Using the chacl Command

Suppose your user name is `leslie`, and you are in the group `users`. If you need to limit access to `myfile` so anyone in your group can read the file, but *only* you and your manager (`arnie`) can both read from and write to the file, you might follow these steps:

1. First, use the `chmod` command to protect `myfile` so your group can read it, but only you can both read from and write to the file. (*Remember:* If you have any previously set ACL entries, `chmod` deletes them unless you use the `-A` option):

   ```
   $ chmod -A 640 myfile
   ```

2. To view the permissions you just set, run the `ll` command on `myfile`:

```
$ ll myfile
-rw-r-----   1 leslie   users          236 Dec  8 14:04 myfile
```

The `ll` command shows that *owner* (`leslie`) has read and write permission on `myfile`; *group* (`users`) has only read permission; *other* has no access to the file.

3. Now use the `lsacl` command to compare the long listing above with the ACL entries:

```
$ lsacl myfile
(leslie.%,rw-)(%.users,r--)(%.%,---) myfile
```

The `lsacl` command shows that `leslie` in any group (`%`) has read and write permission to `myfile`; anyone (`%`) in the group `users` has read permission to `myfile`; and everyone else has no access to `myfile`.

4. To specify that your manager, who is in a different group (`mtg`), should have read and write access to `myfile`, use the `chacl` command to create a new ACL entry for `myfile`:

```
$ chacl 'arnie.mtg=rw' myfile
```

5. Now run the `ll` command on `myfile`:

```
$ ll myfile
-rw-r-----+  1 leslie   users          236 Dec  8 14:04 myfile
```

The plus sign (`+`) at the end of the permissions string means that additional permissions (in the form of optional ACL entries) exist for `myfile`.

6. Run the `lsacl` command to view these optional ACL entries:

```
$ lsacl myfile
(arnie.mtg,rw-)(leslie.%,rw-)(%.users,r--)(%.%,---) myfile
```

The `lsacl` command now shows that, in addition to the previous ACL entries, `arnie` of the group `mtg` has read and write permission to `myfile`.

The table below further illustrates ways you can use the `chacl` command:

If you want to ...	Use this command ...
Create a new ACL entry allowing the user `cyc` in any group (`%`) read and write (`=rw`) access to `myfile`.	`$ chacl 'cyc.%=rw' myfile`
Modify an existing ACL entry allowing all users (`%`) in all groups (`%`) read (`+r`) access to `foofile`.	`$ chacl '%.%+r' foofile`
Modify an existing ACL entry denying all users (`%`) in the `adm` group write (`-w`) access to `afile`.	`$ chacl '%.adm-w' afile`
Create a new ACL entry denying the user `jon` in the `mkt` group read, write, or execute/search access to the `oldir` directory.	`$ chacl 'jon.mkt=' olddir`

For More Information ...

To learn more about security in general, see the *HP-UX Security Manual.*

To learn more about using the `lsacl` command, see the *lsacl* entry in section 1 of the *HP-UX Reference.*

To learn more about using the `chacl` command, see the *chacl* entry in section 1 of the *HP-UX Reference.*

For additional information on ACLs, see the *acl* entry in section 5 of the *HP-UX Reference.*

Index

H

I

K

L

M

T

U

V

W

SYNTAX FOR REDIRECTION AND PIPES

To Do This ...	Use This Command ...
Read standard input from an existing file.	*command* < *infile*
Write standard output to a file.	*command* > *outfile*
Append standard output to a file.	*command* >> *outfile*
Write standard output and standard error to different files in the Bourne and Korn Shells.	*command* > *outfile* 2> *errfile*
Write standard output and standard error to different files in the C Shell.	(*command* > *outfile*) >& *errfile*
Send the output of one command as input to another command using a pipe.	*command1* | *command2*

HP Part Number
98594-90006
Microfiche No. 98594-99006
Printed in U.S.A. E0989

98594-90609
For Internal Use Only