SCO

SCO OpenServer™ Operating System Tutorial

SCO OpenServer™

SCO OpenServer™
Operating System Tutorial

Document Version: 5.0
1 May 1995

About this book

Welcome to the SCO OpenServer™ system which encompasses an operating system based on UNIX technology. For information on where the rest of the product is documented refer to "Related documentation" (page 2).

This book is aimed at people new to the UNIX system who will be working at the command line prompt rather than using the graphical interface. If you have little or no computer experience, relax. This book will introduce you step-by-step to some of the key features of the UNIX system. If you have used the UNIX system before, you can use this book as a refresher or a quick reference guide.

Before you begin, you should have a user account set up for you. Ask your system administrator (the person who looks after your system) to make sure your account is set up.

DOS users who want to get started right away may want to turn to Appendix A, "Going from DOS to UNIX" (page 99). This appendix contains Table A-1, "Equivalent UNIX and DOS commands" (page 100) showing common DOS commands and their UNIX system counterparts.

Although we try to present information in the most useful way, you are the ultimate judge of how well we succeed. Please let us know how we can improve this book (page 5).

How this book is organized

Each chapter in this book is a self-contained lesson for you to work through at the computer. Each lesson builds on what you have learned before.

Question-and-answer sections often follow examples in this tutorial. These sections give you a little more information about what you have learned, and they tell you what to do if you see an error message. Question-and-answer sections are indicated by **Q** and **A** in the margin.

A "Summary" section appears at the end of every chapter. This section summarizes the commands presented in the chapter and tells you where to look in the SCO OpenServer documentation to find more information.

Related documentation

SCO OpenServer systems include comprehensive documentation. Depending on which SCO OpenServer system you have, the following books are available in online and/or printed form. Access online books by double-clicking on the Desktop **Help** icon. Additional printed versions of the books are also available. The Desktop and most SCO OpenServer programs and utilities are linked to extensive context-sensitive help, which in turn is linked to relevant sections in the online versions of the following books. See "Getting help" in the *SCO OpenServer Handbook.*

NOTE When you upgrade or supplement your SCO OpenServer software, you might also install online documentation that is more current than the printed books that came with the original system. In particular, the new information provided online with our regular Advanced Hardware Supplements (AHS) supersedes and frequently obsoletes the material in the printed version of this book. For the most up-to-date information, check the online documentation.

Operating System User's Guide
: provides an introduction to SCO OpenServer command-line utilities, the SCO Shell utilities, working with files and directories, editing files with the **vi** editor, transferring files to disks and tape, using DOS disks and files in the SCO OpenServer environment, managing processes, shell programming, regular expressions, **awk**, and **sed**.

Operating System User's Reference
contains the manual pages for user-accessible operating system commands and utilities (section C).

Release Notes
contain important late-breaking information about installation, hardware requirements, and known limitations. The *Release Notes* also highlight the new features added for this release.

SCO OpenServer Handbook
provides the information needed to get your SCO OpenServer system up and running, including installation and configuration instructions, and introductions to the Desktop, online documentation, system administration, and troubleshooting.

Operating System Administrator's Reference
contains the manual pages for system administration commands and utilities (section ADM), system file formats (section F), hardware-specific information (section HW), miscellaneous commands (section M), and SCO Visual Tcl™ commands (section TCL).

System Administration Guide
describes configuration and maintenance of the base operating system, including account, filesystem, printer, backup, security, UUCP, and virtual disk management.

Graphical Environment Guide
describes how to customize and administer the Graphical Environment, including the X Window System™ server, the SCO® Panner™ window manager, the Desktop, and other X clients.

Graphical Environment help
provides online context-sensitive help for Calendar, Edit, the Desktop, Help, Mail, Paint, the SCO Panner window manager, and the UNIX® command-line window.

Graphical Environment Reference
contains the manual pages for the X server (section X), the Desktop, and X clients from SCO and MIT (section XC).

Guide to Gateways for LAN Servers
describes how to set up SCO® Gateway for NetWare® and LAN Manager Client software on an SCO OpenServer system to access printers, filesystems, and other services provided by servers running Novell® NetWare® and by servers running LAN Manager over DOS, OS/2®, or UNIX systems. This book contains the manual pages for LAN Manager Client commands (section LMC).

Mail and Messaging Guide
describes how to configure and administer your mail system. Topics include **sendmail**, MMDF, **SCO Shell Mail**, **mailx**, and the Post Office Protocol (POP) server.

Networking Guide
provides information on configuring and administering TCP/IP, NFS®, and IPX/SPX™ software to provide networked and distributed functionality, including system and network management, applications support, and file, name, and time services.

Networking Reference
contains the command, file, protocol, and utility manual pages for the IPX/SPX (section PADM), NFS (sections NADM, NC, and NF), and TCP/IP (sections ADMN, ADMP, SFF, and TC) networking software.

PC-Interface Guide
describes how to set up PC-Interface™ software on an SCO OpenServer system to provide print, file, and terminal emulation services to computers running PC-Interface client software under DOS or Microsoft® Windows™.

Performance Guide
describes performance tuning for uniprocessor, multiprocessor, and networked systems, including those with TCP/IP, NFS, and X clients. This book discusses how the various subsystems function, possible performance constraints due to hardware limitations, and optimizing system configuration for various uses. Concepts and strategies are illustrated with case studies.

SCO Merge User's Guide
describes how to use and configure an SCO® Merge™ system. Topics include installing Windows, installing DOS and Windows applications, using DOS with the SCO OpenServer operating system, configuring hardware and software resources, and using SCO Merge in an international environment.

SCO Wabi User's Guide
describes how to use SCO® Wabi™ software to run Windows 3.1 applications on the SCO OpenServer operating system. Topics include installing the SCO Wabi software, setting up drives, configuring ports, managing printing operations, and installing and running applications.

The SCO OpenServer Development System includes extensive documentation of application development issues and tools.

Many other useful publications about SCO systems by independent authors are available from technical bookstores.

Typographical conventions

This publication presents commands, filenames, keystrokes, and other special elements in these typefaces:

Example:	Used for:
lp or **lp**(C)	commands, device drivers, programs, and utilities (names, icons, or windows); the letter in parentheses indicates the reference manual section in which the command, driver, program, or utility is documented
/new/client.list	files, directories, and desktops (names, icons, or windows)
root	system, network, or user names
filename	placeholders (replace with appropriate name or value)
⟨Esc⟩	keyboard keys
`Exit program?`	system output (prompts, messages)
yes or **yes**	user input
"Description"	field names or column headings (on screen or in database)
Cancel	button names
Edit	menu names
Copy	menu items
File ⇨ Find ⇨ Text	sequences of menus and menu items
$HOME	environment or shell variables
"adm3a"	data values

How can we improve this book?

What did you find particularly helpful in this book? Are there mistakes in this book? Could it be organized more usefully? Did we leave out information you need or include unnecessary material? If so, please tell us.

To help us implement your suggestions, include relevant details, such as book title, section name, page number, and system component. We would appreciate information on how to contact you in case we need additional explanation.

To contact us, use the card at the back of the *SCO OpenServer Handbook*, or write to us at:

Technical Publications
Attn: CFT
The Santa Cruz Operation, Inc.
PO Box 1900
Santa Cruz, California 95061-9969
USA

or e-mail us at:

techpubs@sco.com or ... *uunet!sco!techpubs*

Thank you.

Chapter 1

Getting started

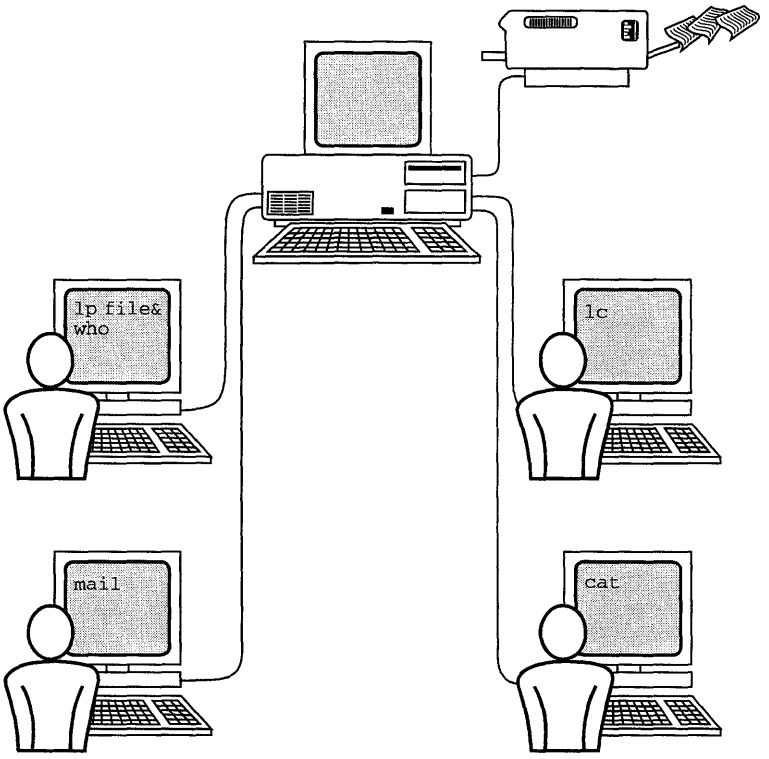
In this chapter, you will learn how to start a work session on a UNIX system (how to log in) and how to finish a work session (how to log out).

Before you begin, you need to know your login name (username), password, and terminal type. Ask your system administrator for this information.

SCO operating system

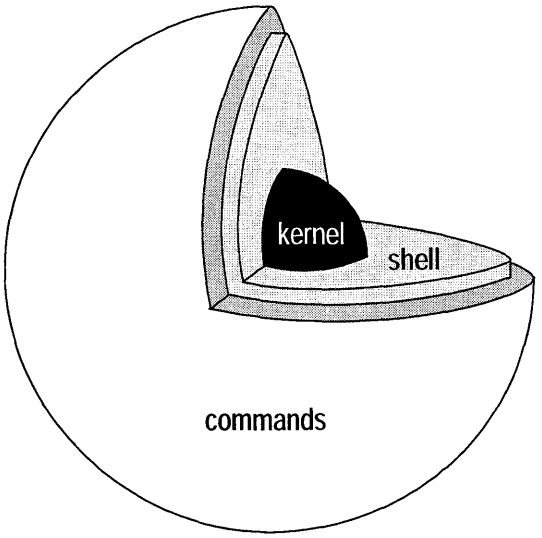
The UNIX operating system is a multiuser, multitasking operating system.

An "operating system" is a program that manages the resources of the computer. An operating system sets up a consistent way for programs to request resources, such as time on the processor, or space in memory, from the computer itself. Operating systems look after all the devices attached to the computer, such as printers, modems, disks, and terminals. Another part of an operating system's job is to maintain a filesystem; that is, to set up a consistent way for information to be stored and retrieved.



On a multiuser, multitasking system, several people can do several tasks at once using the same computer.

The term "the UNIX operating system" usually refers to the kernel, which is the heart of the operating system. People use a variety of shell programs to communicate with the kernel, which, in turn, communicates with the hardware. The UNIX operating system also includes a wide range of programs that meet the day-to-day needs of computer users and programmers.



Three layers of the UNIX system: kernel, shell, and commands

The UNIX system is called a *multiuser* operating system because more than one person can use the computer at the same time. In a typical office setup, one computer runs the UNIX operating system and several people share this computer, each using a terminal which is connected to it.

The UNIX system is called a *multitasking* operating system because each user can do several tasks at once. On a single-tasking operating system, such as DOS, if you type a command that takes a long time for the computer to process, you have to wait for the computer to finish processing before you can continue working. On a UNIX system, you can put commands "in the background." This means you can start working on something else while the computer continues to process your other commands in the background.

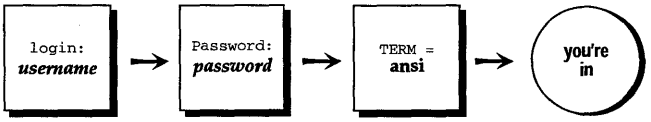
Logging in

To start working on a computer running the UNIX system, the first thing you need to do is log in. When you log in, you tell the computer your name and your password, and it checks them against its records. If everything matches up, the computer starts a login shell for you, puts you in your home directory, and shows you a command prompt. You can then start working on the computer, typing commands at the prompt.

After you turn on your computer or terminal, before you log in, you should see a `login:` prompt:

```
login:
```

This is where you type your login name, which is the name by which the computer knows you.



After you type your login name, you may see a `password:` prompt:

```
login:susannah ⟨Enter⟩
Password:
```

Most UNIX system users have a password that lets them into the computer. If your UNIX system uses passwords, you should keep your password secret so other people cannot use your account without your knowledge.

You should change the password given to you by your system administrator so you have a new password that only you know. On some systems, you may be prompted to change your password as soon as you log in. See "Changing your password" (page 13) for instructions.

If you have correctly typed your login name and password, the computer logs you in. Depending on your system, you may see a "message of the day."

Try logging in now:

1. Switch on your computer or terminal.
2. Press ⟨Enter⟩ a couple of times, until you see the `login:` prompt.
3. Type your login name and press ⟨Enter⟩.
4. If you see the `password:` prompt, type your password and press ⟨Enter⟩.

The computer logs you in. If this is the first time you have logged in, you may be prompted to change your password. You may see some messages about your system, and you may see a prompt for your terminal type. If you are working on a system that has a graphical front-end you will need to open a UNIX window by double clicking on the UNIX icon.

Q: What if I make a mistake typing my login name or password?

A: Press the ⟨Bksp⟩ key to backspace over the misspelling and then retype. You can backspace when you type your password even though you cannot see the letters on the screen.

Q: What if I see the message `Login incorrect`?

A: This means you made a mistake typing your login name or password. Try again at the next `login:` prompt.

(Even if the mistake was in typing your login name, the computer waits until you type your password to tell you. This means if someone is trying to guess your login name or password, they will not know which one they got wrong.)

On some systems, you may see the message `Waiting for login retry:..` between `login:` prompts. These systems keep track of how many times you've tried to log in.

Q: What if I see the message `Login timed out`?

A: Some systems keep track of how many times you tried to log in and, after a certain number of tries, they "time out" the login. This is a security feature to make it more difficult to guess passwords.

If you see a message like this, ask your system administrator to modify your login information so you can try to log in again.

Q: What if everything I type is coming out in capital letters?

A: The UNIX system is sensitive to the difference between upper- and lowercase letters, even at login time. If everything you type appears in capitals, you cannot log in. If there is a ⟨CapsLock⟩ key on your terminal or computer, try pressing it. If this does not work, try switching your computer off and on again. If you still cannot get lowercase letters, ask your system administrator for help.

Your terminal type

After you log in, the computer may ask for your terminal type. This tells the computer running the UNIX system what kind of terminal you are working from, so it can display things in a way your terminal understands.

After you log in, you may see a terminal type prompt similar to this:

```
TERM = (ansi)
```

The terminal type in parentheses is what the computer thinks your terminal should be. If this information is correct, you can press ⟨Enter⟩ to accept it. Otherwise, you should enter the correct terminal type and press ⟨Enter⟩.

In the example above, if you really were working from an *ansi* terminal, you would press ⟨Enter⟩. If your terminal was a Wyse60, instead of an *ansi* terminal, you would enter:

```
TERM = (ansi)wy60 ⟨Enter⟩
```

After you have set your terminal type, you see a command prompt.

Try setting your terminal type now:

1. Log in.
2. If you see the `TERM=` prompt, press ⟨Enter⟩ to accept the terminal type shown in parentheses, or type in the correct terminal type. (If you do not know your terminal type, ask your system administrator.)

Q: What if I do not see a terminal type prompt?

A: Some systems are set up to know what kind of terminal you are logging in on automatically. These systems may not prompt you for the terminal type.

Q: What if I make a mistake and set up the wrong terminal type?

A: If you make a typing mistake, you can backspace to correct it, if you have not pressed ⟨Enter⟩ yet. Otherwise, if you accidentally set up the wrong terminal type, or if you discover while you are working that you have the wrong terminal type, there are two things you can do:

- Log out and log back in again, then choose the correct terminal type.
- Reset your terminal from the command line. The way you reset your terminal in the middle of your work session depends on the shell you are using. (See "Identifying your shell" (page 14) for instructions on determining which shell you are using.) If you are using the Bourne shell (**sh**) or the Korn shell (**ksh**), type:

  TERM=*termtype* ; export TERM

  Here ***termtype*** is the correct terminal type. If you are using the C shell (**csh**), type:

  setenv TERM *termtype*

Changing your password

Depending on how your system is set up, you may be prompted to change your password immediately when you first log in. This is a security feature to ensure that you are the only person who knows your password.

If your system requires you to change your password the first time you log in, you see a message like `Your password has expired.` The computer then starts the password program.

You see a message like `Setting password for user:` ***loginname***, where ***loginname*** is your own login name. Next, you are prompted for your old password. Type the password given to you by your system administrator, then press ⟨Enter⟩. The computer responds with a message like `Password change is forced for` ***loginname***, where ***loginname*** is your login name. You then see the first screen of the password program. Follow the instructions on the screen to pick your own password or to have the computer generate a password for you. Once you successfully change your password, the computer finishes logging you in.

If you want to change your password again later, you can use the **passwd** command to start the password program again. To use the **passwd** command, type **passwd** and press ⟨Enter⟩. You will be prompted for your old password, then you will be given the choice of picking your own password or choosing a machine-generated password. Note that some systems may restrict you from using the **passwd** program at certain times. This is a security feature that allows the system administrator to control how often users change their passwords.

Q: What if I type my old password incorrectly?

A: If you type your old password incorrectly and press ⟨Enter⟩, the password program will terminate. If you are changing your password at login, the computer may log you out. In this case, log in again and carefully type your old password when you are prompted.

If you are still logged in after mistyping your old password, you can just restart the password program by typing **passwd** and pressing ⟨Enter⟩.

Q: What if I forget my password?

A: If you forget your password completely, all is not lost. Tell your system administrator that you have forgotten your password. He or she will be able to modify your records on the computer so you can log in again and choose a new password.

Identifying your shell

Once you have set your terminal type, the computer shows you a command prompt. This is where you type commands during the rest of your work session. Each time you press ⟨Enter⟩, you see a new command prompt. The prompt you see depends on the login shell you are using.

All the time that you are working on the UNIX system, you are working within a shell. When you log in, you are automatically placed within a shell; this is called your login shell. Shells are both command interpreters and programming languages. Each command line you type is interpreted by the shell, which passes your requests to the appropriate program for processing.

For most of this tutorial, you will be using shells only as command interpreters. In other words, you will be typing commands at the prompt and seeing what they do.

In Chapter 9, "Customizing your environment" you will be introduced to the concept of a shell script. These are text files that contain shell language programs.

There are three shells distributed with your SCO OpenServer system:

- Korn shell (**ksh**)
- Bourne shell (**sh**)
- C shell (**csh**)

See Chapter 10, "Configuring and working with the shells" in the *Operating System User's Guide* for information about the differences among these shells and the wide variety of features each shell provides.

By default (unless someone has changed it), the Bourne and Korn shells show a dollar sign ($) as a command prompt. The C shell shows a percent sign (%) by default.

(You can change your prompt by editing a file that the computer reads when you log in. See Chapter 9, "Customizing your environment" for instructions.)

If you cannot tell which shell you are using from the prompt, you can ask the computer by typing **echo $SHELL**. This says, "tell me the value of the variable **SHELL**." The computer responds with an answer like:

```
/bin/sh
```

The last part is the name of the shell, **sh** (the Bourne shell), and the first part is the directory in which it lives.

NOTE If you are running multiple shells the value of the **SHELL** variable will not necessarily reflect the current shell and so this technique can only be relied upon when you first log in.

Logging out

When you have finished using the computer, you should log out.

When you log out, no one can use your terminal until they correctly log in by typing a valid login name and password. Logging out protects you from other people doing potentially destructive things with your files if they are logged in as you. It is a good security practice.

The command you type to log out depends on the shell you are using. To log out using the Bourne shell, type **exit** and press ⟨Enter⟩.

To log out using the C shell, type **logout** then press ⟨Enter⟩.

You may also be able to log out using a quick ⟨Ctrl⟩**D** However, this may be disabled on your system.

When you log out, the `login:` prompt reappears on your screen.

Try logging out now:

1. If you are using the Bourne or the Korn shell, type **exit** and press ⟨Enter⟩. If you are using the C shell, type **logout** and press ⟨Enter⟩.
2. The UNIX system should log you out and the `login:` prompt should reappear on your screen.

Q: What if I see a message like:

```
exir: not found
```

A: In this example, the computer is telling you it cannot find a command named **exir**. What you meant, however, was "exit". Try typing the command again and press ⟨Enter⟩.

Q: What if I try to log out with ⟨Ctrl⟩**D** and I see a message like:

```
Enter "exit" to logout
```

A: Some systems are set up so that you cannot log out with ⟨Ctrl⟩**D**. This is so that people do not accidentally log themselves out when they are typing ⟨Ctrl⟩**D** for another reason. Follow the instructions on the screen to log out correctly.

Q: What if I change my mind when I am typing a command, and I want to cancel the command and start again?

A: There are two ways you can cancel the command you are currently typing and start over:

- press the ⟨Del⟩ key on the numeric keypad

  The ⟨Del⟩ key is called the interrupt key. You can use ⟨Del⟩ to interrupt a command that has started to run as well as to cancel a command you have not yet run. In most cases, this cancels the command and gives you a new prompt. (With some commands, you may need to press ⟨Ctrl⟩**D** or a different quit command.)

- press ⟨Ctrl⟩**U**

  Pressing ⟨Ctrl⟩U discards what you have typed on that line.

Summary

To log in	`login:` ***login name*** `Password:` ***password***
To log out	**exit** *or* **logout** *or* ⟨Ctrl⟩**D**
To set your terminal type	`TERM=(ansi)` ***yourtermtype***
To change your password	**passwd**
To find out what shell you are using	**echo $SHELL**

For more information about	See
what happens when you log in	**login**(M)
changing your password	**passwd**(C)
selecting a password	Chapter 9, "Using a secure system" in the *Operating System User's Guide*

Chapter 2

Electronic mail

In this chapter, you will learn how to send electronic mail and how to read and reply to mail that has been sent to you.

Before you begin, you should know how to log in and how to type commands at the prompt.

Sending mail

You can send electronic mail with the **mail** command to anyone who uses your computer system. If your system has a modem and the UUCP (UNIX-to-UNIX Communications Protocol) program is set up, you can send mail to anyone who is linked to your computer network. To send mail to someone on your system, you type **mail** and then the receiver's login name. For instance, if you wanted to send mail to Doug, you would type **mail doug**. Depending on how **mail** is set up on your system, you may be prompted for a subject; this subject shows up in the list of messages in the receiver's mailbox.

```
$ mail doug
Subject: My promotion
```

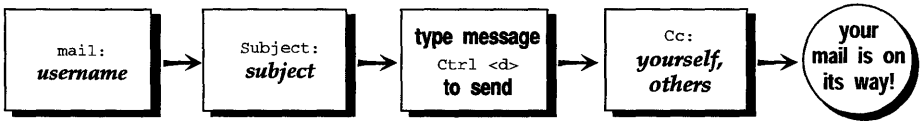
Then, you type your message.

To begin new lines in a logical way, press ⟨Enter⟩ for each new line.

If you make mistakes while you are typing a mail message, you can backspace to correct them. However, you cannot backspace across more than one line. If you want to correct mistakes anywhere in a message, you need to use an editor such as **vi** (the visual editor) to type your message. See "Using the vi editor in mail" (page 26) for instructions.

When you have finished, go to a new line and press ⟨Ctrl⟩**D**. This tells the **mail** program you are ready to send the message. It may show you a `Cc:` prompt; this is where you can type the names of people who you want to receive a "carbon copy" of the message:

```
$ mail doug
Subject: My promotion
Thank you for your recent letter of promotion.
I look forward to the challenge of an executive position,
despite its long hours and tiring international travel.

⟨Ctrl⟩D
Cc: susannah
```



Many people use the `Cc:` prompt to send a copy of their message back to themselves, in addition to copying others. If you do not want to copy anyone, press ⟨Enter⟩ at the `Cc:` prompt.

You do not have to conclude a message with your name; your login name is automatically displayed at the beginning of the message and in the recipient's mailbox.

Some people use **mail** to send reminder messages to themselves.

Try sending a mail message to yourself:

1. Log in.
2. Type **mail** and your login name and press ⟨Enter⟩.
3. If you see a `Subject:` prompt, type **Grocery list** as the subject, and press ⟨Enter⟩.

4. Type in the message, as shown in the following screen display.
5. When you have typed the message, press ⟨Enter⟩ to go to a new line, then press ⟨Ctrl⟩**D**.
6. If you see a `Cc:` prompt, press ⟨Enter⟩. **mail** sends the message, and returns you to the prompt.

```
$ mail susannah
Subject: Grocery list
cat food
dry cat food
flea spray
litter tray liners
tuna
milk
⟨Ctrl⟩D
Cc: ⟨Enter⟩
(end of message)
```

Q: What if I am in the middle of typing a message and I change my mind about sending it?

A: You can cancel a mail message by pressing ⟨Del⟩ twice. The first time you press ⟨Del⟩, **mail** responds with:

```
(Interrupt -- one more aborts message)
```

(This interrupt message does not actually appear in your mail message.)

The second time you press ⟨Del⟩, **mail** cancels the message and returns you to the prompt.

Reading mail

When you first log in, your shell tells you if you have mail with a message like `You have mail`. Depending on how your system is set up, you may also see a message like `You have new mail` when new mail arrives.

To read your mail, type **mail** and press ⟨Enter⟩. If there are no messages in your mailbox, you see a message like:

`No mail for` ***yourloginname***

(***yourloginname*** is really your login name.)

Otherwise, you are brought into your mailbox.

An electronic mailbox lists all the messages you have waiting, and tells you a bit about each:

```
SCO System V Mail (Version 3.2)  Type ? for help.
3 messages:
  >3 susannah Wed Jun 29 15:23  9/237  "Grocery list"
   2 doug     Wed Jun 29  9:00  28/863 "Promotion"
   1 sylvain  Wed Jun 29  8:59  15/391 "Meet you after work"
&
```

In this example, there are three messages waiting, including the practice message from the previous section. Here, the most recent message appears first in the list, but your mailbox may show the oldest message first.

The & is the **mail** prompt. This is where you type commands while you are in **mail**.

The > to the left of a message marks the current message. Next comes the message number; this is how you identify a particular message. The sender, the date and time the message arrived, the number of lines and characters in the message, and as much of the "Subject:" line as can fit are also shown.

To read a message, type its message number, and press ⟨Enter⟩. If the message is too long to fit on one screen, you see a ? at the bottom of your screen. Press ⟨Enter⟩ to see the next page of your message. Once you read a message, it is automatically saved to a file called *mbox*.

You can have another look at the message headers after you have begun to read your mail by pressing **h** (headers) and ⟨Enter⟩. (If you want to stop reading a long mail message to look back at the message headers, press ⟨Del⟩. This interrupts the current message and gives you the main mail prompt _ where you can type **h** and press ⟨Enter⟩ to display the headers.) If you have more than one screen of headers, you can use the **z** command to move forward and backward through the header screens. Type **z** to move to the next screen of headers or **z-** to go back to the previous screen of headers.

To quit **mail**, type **q** and press ⟨Enter⟩ at the main mail prompt.

Try reading the message you sent to yourself:

1. Type **mail** and press ⟨Enter⟩ to enter your mailbox.
2. Type the number to the left of the message you sent yourself and press ⟨Enter⟩. The message is displayed on your screen.
3. Type **q** and press ⟨Enter⟩ at the mail prompt to quit **mail**.

Responding to mail

You can respond to a mail message with the **r** command. If you have just read a message, typing **r** at the mail prompt starts a response to that message. If you have not read any messages, pressing **r** begins a response to the message at the bottom of your screen. You can also type the following at the mail prompt to respond to the message numbered ***number***:

r ***number***

When you respond to a message, **mail** automatically fills in the "To:" field with the name of the sender of the original message, and the "Subject:" field with "Re: ***the original subject***".

Lowercase **r** responds only to the sender of the message. If you want to respond to the sender and everyone who was copied on the original mail, use uppercase **R**.

For practice, send yourself another mail message and then respond to it:

1. Type **mail** and your login name, then press ⟨Enter⟩.
2. At the `Subject:` prompt, type **Test message** and press ⟨Enter⟩.
3. Type **This is a test message** as the body of the message.
4. Go to a new line and press ⟨Ctrl⟩**D** to end the message.
5. If you see a `Cc:` prompt, press ⟨Enter⟩.
6. Go into your mailbox by typing **mail**, then pressing ⟨Enter⟩.
7. At the mail prompt, type the number of the test message you just sent and press ⟨Enter⟩.
8. At the next mail prompt, type **r**.
9. Type a brief response, as shown in the following screen display, then go to a new line, and press ⟨Ctrl⟩**D**.

10. If you see a `Cc:` prompt, press ⟨Enter⟩.

11. To read the response you just sent, type **mail** and press ⟨Enter⟩ to enter **mail** again and then type the number next to the response message. You will be able to tell it is a response to the previous message because it will have the subject of that message in its "Subject:" line. (If you do not see your response immediately, do not worry. It may take a moment or two for your message to arrive.)

12. When you finish, type **q** and press ⟨Enter⟩ to quit **mail**.

```
From susannah Wed Jun 29 15:23:01 1994
To: susannah
Subject: Test message
Date: Wed Jun 29 15:23:02 1994

This is a test message

r
To: susannah
Subject: Re: Test message

Received the test message.
Thanks.

⟨Ctrl⟩D
Cc: ⟨Enter⟩
q
```

More mail features

The **mail** program has a wide variety of features for sending and reading mail. Only a few of these features are explained here; see **mail**(C) for a complete list.

Getting help

mail has two screens of online help that show you the available commands.

To get help when you are typing a message, go to a new line and type ~? (tilde-question mark). This is called a "mail compose escape;" you use these escapes to give a command to the **mail** program while you are typing a message. The help screen shown lists all the **mail** compose escapes.

To get help when you are reading your mail, type ? at the mail prompt. The help screen you see shows the commands available when you are reading your mail.

Saving mail

You can save a mail message you have just read in a file by typing **s** and the name of the file you want to save the message in, then pressing ⟨Enter⟩. You can save any mail message in a file by typing **s**, the number of the message, and the name of the file you want to save it in, and pressing ⟨Enter⟩.

For example, if you want to save the message "Promotion" in a file called *fromdoug*, you could type **s 2 fromdoug**:

```
SCO System V Mail (Version 3.2)  Type ? for help.
3 messages:
  >3 susannah Wed Jun 29 15:23  9/237  "Grocery list"
   2 doug     Wed Jun 29  9:00  28/863 "Promotion"
   1 sylvain  Wed Jun 29  8:59  15/391 "Meet you after wo"
&s 2 fromdoug
"fromdoug" [New file] 28/863
```

This creates a new file called *fromdoug* that contains the mail message from Doug. You can save several messages in the same file by using the same filename each time you save a message. The contents of each message you save will be added to the end of the file as you save them.

Once you have a file that contains several messages, you can use the **mail** program to read these messages by typing **mail -f *filename***, where ***filename*** is the name of the file in which you have saved the messages, then pressing ⟨Enter⟩. For example, if you wanted to use the **mail** program to look back at the message from Doug, you could type **mail -f fromdoug**:

```
$ mail -f fromdoug
1 message
 1 doug     Wed Jun 29  9:00  28/863 "Promotion"
-
```

Deleting and recovering mail

You can delete a mail message you have just read by typing **d** and pressing ⟨Enter⟩. If you want to delete the message you have just read and then read the next message, type **dp** and press ⟨Enter⟩. (This deletes this message and prints the next one to the screen.) Delete any message by typing **d**, the number of the message you want to delete, and then pressing ⟨Enter⟩.

Recover the messages you deleted in the current **mail** session by typing **u** (undelete) and pressing ⟨Enter⟩. The deleted messages reappear in your message list. If you want to undelete a particular message, type **u** and the message number of the message you want to undelete. You can only undelete messages you have deleted during the current **mail** session. Once you quit **mail**, the messages you have deleted are gone forever.

Forwarding mail

You can forward mail to other people from within the **mail** program with the **f** command. To forward the message you have just read, type **f** and the login name of the person to whom you want to forward the message at the mail prompt, then press ⟨Enter⟩. You can forward any message by typing **f**, the message number, the login name of the person to whom you want it forwarded, and then pressing ⟨Enter⟩.

Using the vi editor in mail

You can use the **vi** editor to compose your mail messages. To do this, type **~v** (tilde-v) on a new line when you are composing your mail message. This brings you into **vi**. Any text you have already typed appears in **vi**, ready for editing. (For instructions on using **vi**, see "Using the vi editor" (page 43).)

Mailing several people at once: aliases

If you find that you are mailing messages to the same group of people over and over again, you may want to set up a personal mail alias. A mail alias is a single word you substitute for the names of several recipients.

For example, let's say you often found yourself sending messages to all the sales people in your company: Jane, John, Jim, Joe, and Biff. You could create an alias named *sales* so instead of typing **mail jane john jim joe biff**, you could simply type **mail sales**. To set up a mail alias, type:

a *aliasname loginname loginname*...

Here, ***aliasname*** is what you want to call the alias and each ***loginname*** is the login name of a person you want included in the alias. Do this at the mail prompt when you are in your mailbox.

For example, to set up the *sales* alias, you would type the following at the mail prompt when you are reading your mail:

a sales jane john jim joe biff

Summary

To send mail	**mail** ***username***
To read mail	**mail**
To respond to mail	**r** [***messagenumber***]
To use the editor while sending mail	**~v**
To get help while sending mail	**~?**
To get help while reading mail	**?**
To save a message to a file	**s** [***messagenumber***] ***filename***
To cancel a message you are sending	⟨Del⟩ ⟨Del⟩
To look at message headers	**h**
To see the next screen of message headers	**z**
To return to the previous screen of message headers	**z-**
To use the **vi** editor to compose a message	**~v**
To delete a message	**d** [***messagenumber***]
To delete this message and display the next	**dp**
To recover messages deleted within this session	**u** [***messagenumbers***]
To create an alias from within mail	**a** ***aliasnames usernames***
To quit mail	**q**

For more information about	See
Using **mail**	*Mail and Messaging Guide*
All the **mail** commands and options	**mail**(C)

Chapter 3

Directories and files

In this chapter, you will learn how information on a UNIX system is organized into directories and files. You will learn how to move from directory to directory and how to list the files in a directory.

Before you begin, you should know how to log in to your SCO system and how to type commands at the prompt.

The examples in this chapter assume that you are running the Korn shell. If you are not sure which shell you are running see "Identifying your shell" (page 14). To start a temporary Korn shell type **ksh** and press ⟨Enter⟩.

Directories

Information held in a UNIX system is organized in files. Files, in turn, are organized into directories. The directories themselves are organized into a tree structure: that is, there is one common root from which there are branches, from which there are more branches, and so on.

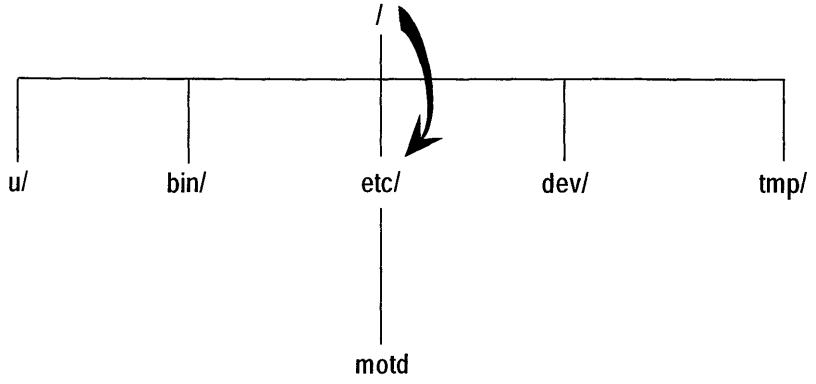
To go to a place on the computer, you need to know its pathname. The pathname tells the computer which directory you want to go to or look at.

An absolute pathname begins with the *root* directory and specifies every directory on the way to the directory or file you want to work with. A relative pathname tells the computer to go to a particular directory relative to the directory where you are right now. Directories are separated by slashes (/) in pathnames. The last word of a pathname is either a directory name or a filename.

This is the pathname for the message of the day, the message that is displayed when you log in to the computer:

/etc/motd

This says "go to the *root* directory, (*/*), then go into *etc*, then go to *motd*." (DOS users will notice that pathnames on the UNIX system are like pathnames on DOS, only the slashes point the other way.)



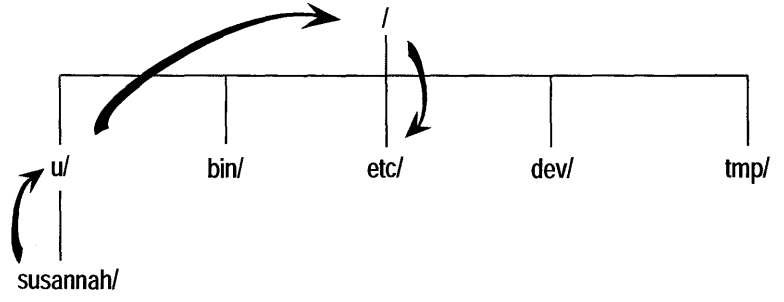
cd /etc

To say "one directory up from here," use the shorthand ".." (dot dot). The shorthand for the directory you are in is "." (dot), although you rarely find reason to type this.

Here is the pathname for */etc/motd*, but shown as a pathname relative to */u/susannah*:

../../etc/motd

This says "go up two directory levels (which takes you to the *root* directory), then go into *etc*, then go to *motd*."



cd ../../etc

If we wanted to go into *Tutorial*, a directory below */u/susannah*, the relative pathname would be *Tutorial*.

Pathnames without a leading " / " are relative pathnames.

Your home directory

When you first log in to the computer, the operating system places you in your home directory. Typically, this has a pathname like either of the following:

*/usr/**loginname***
*/u/**loginname***

Here, ***loginname*** is your login name.

The shorthand for your home directory is **$HOME**. You will see this referred to in the following sections.

Identifying your current directory

To find out the name of your current directory, type **pwd** (print working directory) and press ⟨Enter⟩:

```
$ pwd
/u/susannah
```

Changing directories

To change to a new directory, type **cd** (change directory) and the pathname of the directory you want to change to, then press ⟨Enter⟩:

```
$ pwd
/u/susannah
$ cd /usr/adm
$ pwd
/usr/adm
```

You tell **cd** which directory to change to by giving it an argument. You can use either a relative or an absolute (starting with " / ") pathname as an argument to **cd**.

If you type **cd** with no arguments, you go to your home directory:

```
$ pwd
/etc
$ cd
$ pwd
/u/susannah
```

You can also change to your home directory by saying **cd $HOME**:

```
$ pwd
/usr/lib
$ cd $HOME
$ pwd
/u/susannah
```

Try moving around some directories now:

1. Type **pwd** and press ⟨Enter⟩ to see where you are starting from.
2. Next type **cd /etc** to go to the *etc* directory, one directory down from *root*. (The */etc* directory is where many system administration tools are stored.)
3. Type **pwd** and press ⟨Enter⟩ to check that you are in the right place.
4. Then type **cd default; pwd** and press ⟨Enter⟩. (See the "Q and A" section below for an explanation of the use of " ; " between commands.)
5. Type **cd /usr/spool/lp/requests**. You should see a message like `/usr/spool/lp/requests: Permission denied.` (*/usr/spool/lp/requests* is a directory the computer uses to store printer requests temporarily.)
6. Type **pwd** and press ⟨Enter⟩.
7. You type **cd; pwd** ⟨Enter⟩ to return to your home directory and check that you are there.

```
$ pwd
/u/susannah
$ cd /etc
$ pwd
/etc
$ cd default; pwd
/etc/default
$ cd /usr/spool/lp/requests
/usr/spool/lp/requests: Permission denied
$ pwd
/etc/default
$ cd; pwd
/u/susannah
```

Q: Why do I have to press ⟨Enter⟩ after every command?

A: The carriage return you type at the end of a command line tells the computer to process the command.

Q: What does the " ; " do between two commands?

A: The semicolon (;) is a command separator. It tells the computer that the next word is the start of a separate command, instead of an argument for the previous command.

```
$ cd; pwd
```

This says "change directory, print working directory."

```
$ cd pwd
```

This says "change to the directory named *pwd*."

Semicolons allow you to put more than one command on a line before you press ⟨Enter⟩ to have all the commands processed.

Q: What does the message `Permission denied` mean?

A: The UNIX system uses file and directory permissions to control who can look at, and who can change, files. These permissions are discussed in Chapter 7, "Protecting files and directories" (page 69). When you see the message `Permission denied` it means the permissions on a directory are set so you cannot go into the directory. This is frequently the case for system directories, such as */usr/spool/lp/requests*, and for other people's home directories.

Q: What happens if I misspell a directory name?

A: If you misspell a directory name, the computer may attempt to guess what you meant. Type **y** to accept its guess and change to the directory, or **n** to return to the prompt:

```
$ cd /etv
cd /etc? y
$ pwd
/etc
```

Files

Now that you know how to move from directory to directory, the next step is learning how to see what files are in a directory.

In your SCO OpenServer system, the files that make up the operating system are distributed in special directories called "storage sections" and many of these files are then "symbolically linked" into their normal locations within the filesystem. When a symbolic link is being used, the filename in a directory listing will be followed by an arrow symbol and then the full pathname of where the actual file is located. For example, the file */etc/motd* is actually located in */var/opt/K/SCO/Unix/5.0.0Ca/etc/motd*. This will appear as:

```
motd -> /var/opt/K/SCO/Unix/5.0.0Ca/etc/motd
```

In order to resolve these links so you actually see the files and their attributes and not the links you may want to use the **-L** option to the **l** command.

Listing the files in a directory

There are several different commands you can use to list files. All of these are variations on the **ls** (list) command.

Two common ways of listing files are **ls** and **lc**. **ls** lists files alphabetically in a single column down your screen:

```
$ ls /etc
accton
adfmt
asktime
at.mvw
at.sys
atstart.sys
badtrk
brand
checklist
...
```

lc (list columns) lists files in columns across your screen:

```
$ lc /etc
accton      fd135ds9boot0   logger      opasswd      tpmd.perms
adfmt       fd135ds18boot0  login       passwd       ttys
...
```

lf (list files) is another variation on **ls**. **lf** lists files in columns across your screen, marking programs with a " * ", symbolic links with a " @ " and directories with a " / ":

```
$ lf /etc
accton@         gettydefs@       mkinittab@       siomake@
adfmt@          gettydefs.orig@  mknod@           sioput@
...
```

If symbolic links are present in a directory, using the **-L** option as in the example below identifies the types of file being referred to by the link. The files would look like this:

```
$ lf -L /etc
accton*         gettydefs        mkinittab*       siomake*
adfmt*          gettydefs.orig   mknod*           sioput*
...
```

See **ls**(C) for more information about all the file listing commands.

Try listing the contents of */bin*, a directory where many UNIX commands live:

1. Type **cd /bin** and press ⟨Enter⟩.
2. Type **pwd** and press ⟨Enter⟩ to check where you are.
3. List the directory by typing **lf -L** and pressing ⟨Enter⟩. Your screen should look something like the screen display below.
4. Type **cd** and press ⟨Enter⟩ to return to your home directory.

```
$ cd /bin
$ pwd
/bin
$ lf -L
adb*      dc*       fsck*       mesg*      restor*     tee*
ar*       dd*       getopt*     mkdir*     restore*    telinit*
arV*      df*       gets*       mv*        restorL*    test*
as*       diff*     grep*       ncheck*    restorS*    time*
asm*      diff3*    grpcheck*   newgrp*    rm*         tmp.spx.si
asx*      dircmp*   hd*         nice*      rmdir*      touch*
...
$ cd
```

Hidden files

Files whose names begin with "." (dot) are hidden from view in a normal directory listing. Certain programs, such as **mail** and your shell, create hidden files to avoid cluttering your home directory with unnecessary files. You may want to create hidden files yourself, for example, to store personal mail.

To see hidden files, you need to add the **-a** (all) option to the list command. To see all the files in a directory, you can type **ls -a**, **lc -a**, or **lf -a**.

Try listing all the files in your home directory:

1. See if you are in your home directory by typing **pwd** and pressing ⟨Enter⟩.
2. If you are not in your home directory, type **cd** and press ⟨Enter⟩.
3. List all the files in your home directory by typing **ls -a**.

If you are using the Korn shell, you see something like this:

```
$ ls -a
.
..
.kshrc
.mailrc
.profile
.lastlogin
```

The first two files (" . " and " .. ") are placeholders that refer to the current directory (the one you are in) and its parent directory (the one above it). (Remember, the shortcut for going to the directory above where you are right now is **cd ..**) You see " . " and " .. " in every directory where you list all the files.

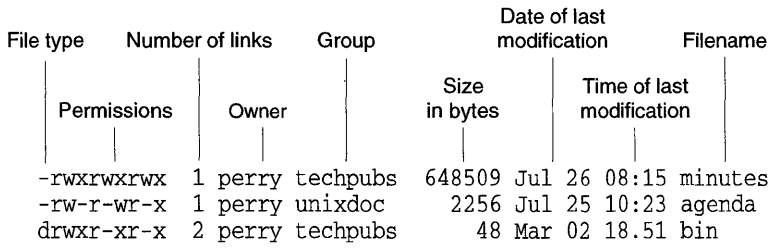
The *.kshrc* and *.profile* files are files that the Korn shell reads when you first log in. These files control your environment: that is, they control the way you work on a UNIX system. For more information, see Chapter 9, "Customizing your environment" (page 87). (Bourne shell users see a file named *.profile*, and C-shell users see a file named *.login* and a file named *.cshrc*; these are the files those shells read at login.) The file *.lastlogin* keeps track of the last time you logged in.

Listing more information about files

So far, you have seen how to list the names of files, and how to see whether files are directories, programs, or regular files. You can use the **-l** (long) option to the **ls** command to see more information:

```
$ ls -l -L /etc
total 4566
-rwx------   1 bin      bin        10510 Mar 16  1993 aiolkinit
-rw-rw-r--   1 root     mem           48 Jun 17 08:34 arp_data
-rwx------   2 bin      bin         2966 Mar 15  1993 asktime
-rwx------   2 bin      bin         2966 Mar 15  1993 asktimerc
drwxrwx--x   4 auth     auth          80 Mar 24  1993 auth
-r-x------   1 bin      auth        1706 Mar 15  1993 authckrc
-rwx--x--x   1 root     root      128266 Mar 12  1993 automount
-rw-r--r--   1 root     root           0 Jun 17 08:34 advtab
...
```

l is another way of saying **ls -l**. A long listing shows you, from left to right, the file type, the permissions on the file, the number of links to the file, the owner of the file, the group of the file, the size of the file in bytes, the date and time the file was last modified, and the name of the file. If a file has not been modified since last year, the year appears instead of the modification time.



What you see in a long listing

Most of this information is discussed in more detail in Chapter 7, "Protecting files and directories" (page 69).

Try doing a long listing of the files in your home directory:

1. Type **cd** and press ⟨Enter⟩ to go to your home directory.
2. Type **l** and press ⟨Enter⟩ to see a long listing of the files.

Q: What if I list a directory and there are so many files that the files at the beginning of the list run off the top of the screen before I have a chance to read them?

A: You can use ⟨Ctrl⟩S and ⟨Ctrl⟩Q to stop and start scrolling output. When you press ⟨Ctrl⟩S, the screen stops scrolling. If you now press ⟨Ctrl⟩Q again, the screen resumes scrolling.

Narrowing the listing: using wildcards

You have seen in the examples in this chapter that sometimes a UNIX system directory has so many files that listing the directory fills more than a screen. If you have some idea of the files you are looking for, you can narrow your search using wildcard characters:

```
$ cd /bin
$ lc c*
cal     cb      chgrp   chown   cmchk   comm    cp      csh
cat     cc      chmod   chroot  cmp     copy    cpio    csplit
```

A wildcard character takes the place of another character or characters. They are also known as metacharacters, because they have a meaning beyond that of a single, regular, character. In the example above, the "*" is a metacharacter, so the command reads: "list all files starting with a "c", followed by any other character or characters." Metacharacters are interpreted by the shell, rather than by commands.

Here are the filename metacharacters:

Metacharacter	Means
*	Any character or characters, including no characters at all
?	Any single character
[...]	Any enclosed character; specify a range with "-"; for example, to match *file.a*, *file.b* or *file.c*, you could use *file.[a-c]*

Here are some more examples:

```
$ cd /etc
$ lc [cde]*
checklist   cron        custom    devnm    dmesg       ext.perms
clri        cshrc       ddate     divvy    dsmd.perms
cmos        cshrc.bak   debrand   dkinit   emulator

default:
archive   cc       format   lock     micnet    passwd    tape.00
archive-  cron     goodpw   login-   mkuser    passwd-   tar
backup    dumpdir  idleout  login    mkuser-   restor    usemouse
boot      dumpsrv  imagen   lpd      msdos     su        xnet
boothd    filesys  lang     mapchan  netbackup tape

$ cd /etc
$ lc [c-e]*
checklist   cron        custom    devnm    dmesg       ext.perms
clri        cshrc       ddate     divvy    dsmd.perms
cmos        cshrc.bak   debrand   dkinit   emulator

default:
archive   cc       format   lock     micnet    passwd    tape.00
archive-  cron     goodpw   login-   mkuser    passwd-   tar
backup    dumpdir  idleout  login    mkuser-   restor    usemouse
boot      dumpsrv  imagen   lpd      msdos     su        xnet
boothd    filesys  lang     mapchan  netbackup tape
```

Both the first and the second example list all the files in */etc* beginning with a "c", "d", or "e" and followed by any other characters, but the second example uses a range [c-e] to do it.

```
$ l /etc/q?
l: /etc/q? not found: No such file or directory (error 2)
```

In the third example, **l /etc/q?** does not produce a list of files because the computer is looking for a file in */etc* that begins with a "q" and has just one other character following it; this does not match any of the files in */etc*. (C shell users would see the message `No match.`)

With the "?" metacharacter, you must type as many ?s as there are letters in the filename you want to match. For example, to search for a six-character filename in */etc* directory which begins with "pa", enter:

```
$ l /etc/pa????
-r--r--r--   1 root     techpubs    2968 Jun 19 15:28 /etc/passwd
```

Try using a metacharacter to find the message of the day file:

1. Type **cd /etc** and press ⟨Enter⟩.
2. Type **l mo*** and press ⟨Enter⟩ to see a long listing of all the files beginning with "mo" in */etc*. (Your screen should look something like the following screen display.) */etc/motd* is the message of the day file.

```
$ cd /etc
$ l mo*
-rw-r--r--   1 root     sys          111 Nov  3 02:34 motd
-rwx--x--x   2 root     bin        27564 Jan  5 04:53 mount
-rwx------   1 root     bin         1071 Nov  3 02:22 mountall
-rwx--x--x   1 root     root       23180 Nov  3 02:32 mountd
```

Summary

To see what directory you are in	**pwd** (print working directory)
To change directories	**cd** ***pathname***
To go to your home directory	**cd** **cd $HOME**
To list files	**ls**
To list files in columns	**lc**
To list files and show type	**lf**
To list all files, even hidden ones	**ls -a**
To make a long listing of all files beginning with "m"	**l m*** or **ls -l m***

For more information about	See
pwd, **cd**, and the **ls** family of commands	**pwd**(C), **cd**(C), and **ls**(C)

Chapter 4

Writing and editing

In this chapter, you will learn how to use the **cat** command to create a file, and how to use the basic features of the powerful **vi** editor. You will also learn how to print files, check on files that are printing, and cancel print requests.

Before you begin, you should know how to log in, and how to enter commands at the prompt. You should know what files and directories are, and what metacharacters are and what they do. You should also know how to start and stop screen scrolling with ⟨Ctrl⟩**Q** and ⟨Ctrl⟩**S**.

Putting text into a file

You can use the **cat** command to create a file quickly by typing **cat > *filename***. **cat** creates a file named ***filename***, and puts the text you type into the file, until you tell it you have finished by pressing ⟨Ctrl⟩**D**. (The name **cat** is short for concatenate, or join together; this is another thing the **cat** command can do.)

```
$ cat > todo
write staff report
review budget figures
return doug's call
⟨Ctrl⟩D
```

Using **cat** to write a file is like writing a mail message: you can backspace to correct mistakes within a line, but you cannot backspace past the beginning of the line you are on. The **vi** editor, discussed later in this chapter, lets you correct mistakes anywhere in a file.

Try writing a file with **cat**:

1. Type **cd; pwd** and press ⟨Enter⟩ to make sure you are in your home directory.
2. Type **cat > mytodo** and press ⟨Enter⟩ to open up a file called *mytodo*. Everything you type now goes into *mytodo*, until you press ⟨Ctrl⟩**D**.
3. Type in the text as shown in the following screen display. Remember to press ⟨Enter⟩ to start each new line.
4. When you have finished typing the text, go to a new line and press ⟨Ctrl⟩**D**.
5. Type **l mytodo** and press ⟨Enter⟩ to check that the file was created.

```
$ cd; pwd
/u/susannah
$ cat > mytodo
write status report
fill out timesheet
buy cat food
⟨Ctrl⟩D
$ l mytodo
-rw-rw----   1 susannah techpubs      52 Jun 24 12:12 mytodo
```

Q: What if I see a message like:
`mytodo: Permission denied` or
`mytodo: cannot create`?

A: When you see a `Permission denied` or `cannot create` message, this means you do not have permission to write in the directory where **cat** is trying to create a file. (File and directory permissions are covered in Chapter 7, "Protecting files and directories" (page 69).) Try changing to your home directory by typing **cd** and pressing ⟨Enter⟩, then try opening up your new file there.

Filenames

Filenames can be up to 256 characters long. You can use any characters you like in a filename, except for the following metacharacters, which have a special meaning to the shell:

! ? * " ' ` ; / $ < > ( ) | { } [ ] ~

It is a good idea to choose meaningful filenames to make it easier to remember what the files contain.

Examples of legal filenames	Examples of illegal filenames
qtrone	qtr(one)
report.127	report | 127
annualreport_1994	annualreport~1994

Filenames only have to be unique within a directory. In other words, you can have as many files named *report* as you like, as long as there is only one *report* per directory.

Looking at files

You can also use the **cat** command to display files on the screen. **cat** sends the whole file to the screen without splitting it into pages. If there are more lines than can fit on the screen, you need to use ⟨Ctrl⟩**S** to control the scrolling.

Try looking at the file you typed with **cat**:

1. Type **cd** and press ⟨Enter⟩ to go to your home directory.
2. Type **cat mytodo** and press ⟨Enter⟩. (You could have typed **cat $HOME/mytodo** instead; this would display the file *mytodo* from anywhere on the system.)

Using the vi editor

vi is the standard text editor on the UNIX system. **vi** is a text editor, not a word processor. It has many powerful features for manipulating text (deleting, moving, searching, replacing, and so on), but it does not, for example, allow you to change line spacing or make letters boldface or italic.

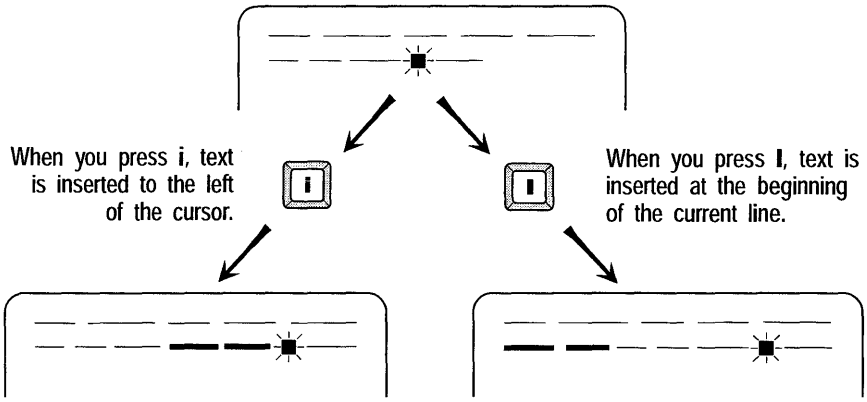
In this tutorial, you will be learning the basics of **vi**. A summary of **vi** commands appears at the back of this book. To appreciate fully the power of this UNIX system tool, you should read Chapter 4, "Editing files" in the *Operating System User's Guide* and look at **vi**(C).

The name **vi** comes from the word "visual." Different from its predecessors **ex** and **ed**, **vi** shows a full screen of the file at once. (**ed** and **ex** are similar to **edlin** on DOS; you tell them what you want to do without actually seeing the file in front of you.)

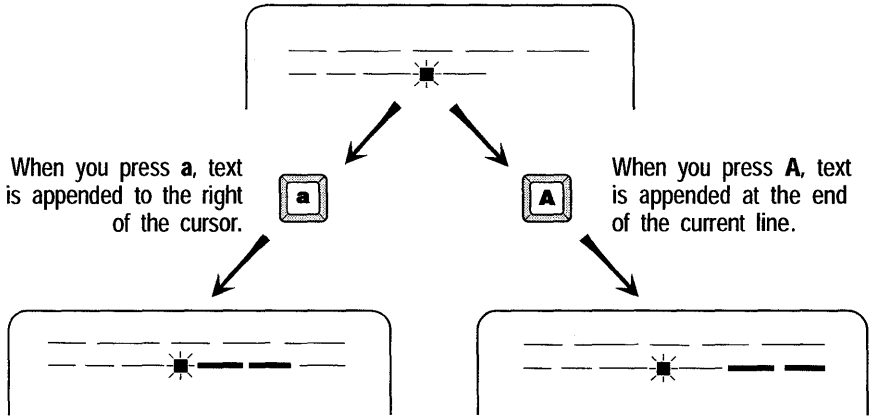
vi works in two modes. When you first start **vi**, you are in *command mode* — **vi** is waiting for you to give it a command. When you give the command **i** (insert), you change into *insert mode*. From then on, what you type is inserted into the file. To leave insert mode and return to command mode, press the ⟨Esc⟩ key. To exit **vi**, give it the command **:x** (you need to press ⟨Esc⟩ first if you are not in command mode). This saves the file if you made any changes.

Entering text

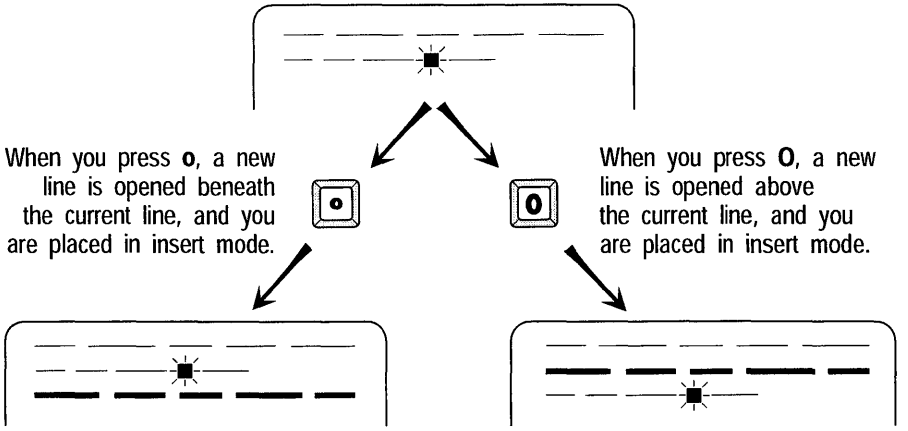
To create a new file in **vi**, type **vi *filename***, where ***filename*** is the name you want to call the new file. (You can edit any existing text file with **vi** by typing **vi** and then the name of that file.)



Inserting text with i or I



Appending text with a or A



Opening a line with o or O

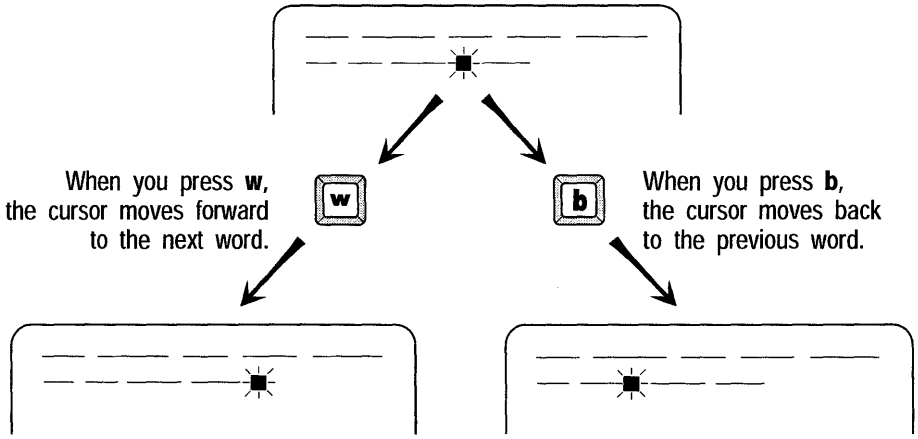
Try writing a file with **vi**:

1. Type **cd** and press ⟨Enter⟩ to go to your home directory.
2. Type **vi weekrep** and press ⟨Enter⟩. This opens up a file called *weekrep* and puts you into the **vi** editor with a blank file in front of you. You can see the name of the file at the bottom of the screen.
3. Type **i** to go into insert mode.
4. Type in the text shown in the following screen capture. Press ⟨Enter⟩ when you want to begin a new line. Do not worry if you make mistakes; you will learn how to correct these shortly.
5. When you have finished typing, press ⟨Esc⟩ to go to command mode.
6. In command mode, type : (colon). You should see a : prompt at the bottom of the screen.
7. Type **x** and press ⟨Enter⟩ at the : prompt. When **vi** writes out the file, it shows you the number of lines and the number of characters.

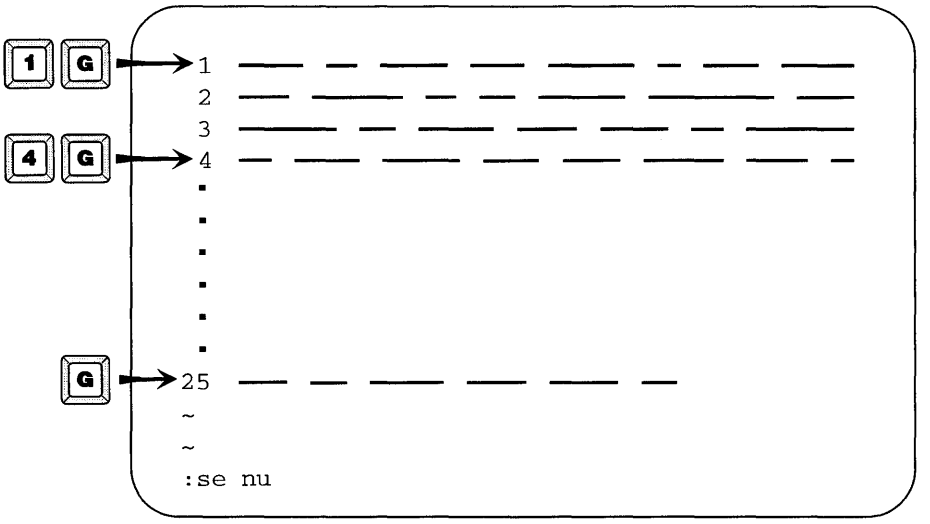
```
Weekly Report

This week, I met with 5 of our 10 distributors.
Everyone is eager to see the next release of our software,
and they all expect to sell a lot of units in the coming quarter.
~
~
~
   .
   .
   .
"weekrep" [New file]
```

Moving around in a file

Use the direction keys (arrow keys) to move one line up or down, or one character right or left. You can also use **h** to move left, **l** to move right, **k** to move up, and **j** to move down.



Moving one word forward (w) or one word back (b)



Moving to a particular line

See Chapter 4, "Editing files" in the *Operating System User's Guide* for some other ways to move around a file in **vi**.

Correcting mistakes

You can correct mistakes in **vi** by using the **x** key to delete the character under the cursor.

You need to be in command mode (press ⟨Esc⟩) to use **x** to delete — otherwise, you just end up with a lot of x's. After you have deleted the unwanted text, insert the correct text by pressing **i**. Text is inserted to the left of the cursor.

Try editing the first line of *weekrep*:

1. Go to your home directory by typing **cd** and pressing ⟨Enter⟩.
2. Type **vi weekrep** and press ⟨Enter⟩ to open *weekrep* in **vi**.
3. Using the down arrow or the **j** key, go to the line starting with "This week ..."
4. Using the right arrow or the space bar, move the cursor to 5 and press **x**. The 5 disappears.
5. Using the right arrow or space bar, move the cursor to 10. Press **x** twice to delete the 10. The line should now read "This week I met with of our distributors."

6. Using the left arrow or the **h** key, move back to the space between "with" and "of". The cursor should be on the space between the two words.
7. Press **i** to insert, then type **6**.
8. Press ⟨Esc⟩. The line should now read "This week I met with 6 of our distributors."
9. Use the right arrow or the space bar to move the cursor to the space between "our" and "distributors".
10. Press **i** and type **11**. The line should now read "This week I met with 6 of our 11 distributors."
11. Press ⟨Esc⟩ to go into command mode, then press : to get the : prompt at the bottom of the screen.
12. Type **x** at the : prompt to save the file and exit **vi**.

Q: I am finding it hard to tell when I am in insert mode and when I am in command mode — is there any way to make this easier?

A: If you are in **vi**, you can press ⟨Esc⟩ to go to command mode, type :, and then type **set showmode** to set the showmode option. The showmode option prints the mode you are in at the bottom of your screen whenever you are in input (insert) mode. The mode it prints will be `APPEND`, `CHANGE`, `INSERT`, `OPEN`, or `REPLACE`, depending on your current action.

If you always want to use the showmode option, create a file in your home directory called *.exrc* that contains the following line:

```
set showmode
```

vi looks for the *.exrc* file each time it starts, so this is where you should put frequently used **vi** options. For more information about **vi** options, see the section on **vi** in Chapter 4, "Editing files" in the *Operating System User's Guide.*

Q: Suppose I type a colon and then change my mind and decide I do not want to use the : prompt?

A: Press ⟨Del⟩ to cancel the command and return to editing the file. Your terminal may beep or flash at you; ignore it. You can also type ⟨Esc⟩ if you have not typed a valid command, but, if you have typed a command and you press ⟨Esc⟩, **vi** performs the command you typed.

Q: Why does my terminal keep beeping (or flashing) at me?

A: **vi** sends a beep to your terminal (some terminals use a flash) in a number of instances. Two common times **vi** beeps at you are when you press ⟨Esc⟩ when you are already in command mode, and when you try to move beyond the last text on a line.

When you first start using **vi**, your terminal beeps a lot. You can safely ignore this.

Printing files

You can use the **lp** (lineprinter) command to print text files on your local printer. To use **lp**, type **lp**, the name of the file, and press ⟨Enter⟩. The computer responds with the number of your print job and a copy of the file is sent to the printer. This works with any kind of printer, not just a lineprinter.

Try printing the weekly report file you were editing:

1. Type **cd** and press ⟨Enter⟩ to go to your home directory.
2. Type **lp weekrep** and press ⟨Enter⟩ to print *weekrep* to your printer.

Printing several copies

Use the **-n** (number) option to **lp** to tell it the number of copies you want to print. To use the **-n** option, type **lp -n*number filename*** where ***number*** is the number of copies you want to print and ***filename*** is the name of the file.

For example, if you wanted to print two copies of your weekly report, you would type **lp -n2 weekrep**

Checking on a print job

When a lot of people are trying to use the same printer at the same time, the print queue can become very long. To see where your job falls in the queue, use the **lpstat** (lineprinter status) command:

```
$ lpstat
tpubs_lw2-8155      root          2147   Jul 29 14:32 on tpubs_lw2
tpubs_lw2-8156      susannah      38884  Jul 29 14:33
tpubs-lw2-8157      susannah      40765  Jul 29 14:38
tpubs-lw2-8158      nigel         24399  Jul 29 14:39
```

lpstat shows all the currently queued print requests. The job ID for each request is shown (for example, tpubs_lw2-8155), with the login name of the

user who issued the print command, the size of the print job in bytes, the date and time of the request, and, if the file is actually on the printer at the time, a message to this effect.

If there are several jobs in the queue, typing **lpstat** shows you all the jobs.

Canceling a print job

To cancel a print job, you need to know its job ID. You can find this out by typing **lpstat** and pressing ⟨Enter⟩, as explained in the preceding section. To cancel a print job, type **cancel**, the ID number of the job, and press ⟨Enter⟩.

Try sending a job to the printer and then canceling it:

1. Type **lp /etc/passwd** and press ⟨Enter⟩ to print the file */etc/passwd*.
2. Type **lpstat** and press ⟨Enter⟩ to check the job ID number of your print job. Write this down.
3. Cancel your print job by typing **cancel**, the job ID you have written down, and press ⟨Enter⟩.
4. Type **lpstat** and press ⟨Enter⟩ to confirm that your job has disappeared from the print queue.

Summary

To create a file with **cat**	**cat** > ***filename***
	text
	⟨Ctrl⟩**D**
To display a file	**cat** ***filename***
To pause/resume the screen scrolling	⟨Ctrl⟩**S**/⟨Ctrl⟩**Q**
To start writing a file with the **vi** editor	**vi** ***filename***
To insert text in **vi**	**i**
To return to command mode from insert mode	⟨Esc⟩
To quit **vi**, saving any changes	**:x**
To move around in **vi**	use the arrow keys or **h** for left, **l** or ⟨Space⟩ for right, **k** for up, and **j** for down
To delete a character in **vi**	**x**
To print a file	**lp** ***filename***
To print 2 copies of a file	**lp -n2** ***filename***
To print a file in the background	**lp** ***filename*** **&**
To run any job in the background	***commandline*** **&**
To check the status of a print job	**lpstat** [***job_id***]
To cancel a print job	**cancel** ***job_id***

For more information about	See
cat	**cat**(C)
vi	Chapter 4, "Editing files" in the *Operating System User's Guide*
	vi(C)
lp, **lpstat**, **cancel**	**lp**(C), **lpstat**(C), and **cancel**(C)

Chapter 5
Managing files

In this chapter, you will learn more about files and directories. You will learn how to use a paging program to read long files screen by screen, and you will learn about two utilities to look at the very top and the very bottom of a file. You will also learn how to make directories, how to remove directories, and how to copy, move, and remove files.

Before you begin, you should know what a file is and what a directory is. You should also know how to tell what directory you are in, how to change directories, and how to list files.

More ways to look at files

In Chapter 3, "Directories and files" (page 29), you learned how to use the **cat** command to display a file on the screen by typing **cat *file***. **cat** simply sends the whole file to the screen; you cannot choose how much of the file you want to see at a time.

Reading a file one screen at a time

The **more** command displays a file one screen at a time. If the file fits on one screen, **more** quits and you are returned to the prompt. If the file is more than one screen long, **more** displays a prompt at the bottom of the screen showing what percentage of the file you have already read. Press ⟨Space⟩ to see another screen of the file. (You can use ⟨Enter⟩ to see another line of the file.) To quit **more** without reading the whole file, type **q**.

The **more** command lets you search for words in a file by typing a slash (/). Type a " / ", then type the word or words you want to search for at the / prompt at the bottom of the screen. **more** skips to the next page of the file where those words occur. **more** can only search forward; if the words you are searching for come before where you currently are in the file, **more** cannot find them.

Try reading through the file */etc/passwd* (a file that stores a variety of information about users on the system) to find your login name:

1. Type **more /etc/passwd** and press ⟨Enter⟩. The first screen of the file will be displayed.
2. Type /, then type your login name at the / prompt at the bottom of the screen and press ⟨Enter⟩.
3. **more** skips to the page where it finds your login name. (This does not work if you have already seen your login name on the screen because **more** cannot search backward.)
4. If you still see a `More` prompt at the bottom of the screen, press **q** to quit **more**.

Q: **more** seems much better than **cat**; is there any reason I should use **cat** instead?

A: **cat** can be better than **more** in some instances. If you want to look at a short file, **cat** is probably better because it does not waste time loading the file into a buffer and then paging it out on the screen. Also, **cat** is more forgiving about what it displays. **more** cannot display files containing control characters (it gives you the error message `Not a text file`), whereas **cat** tries its best to display any file.

Reading just the first or last lines of a file

Sometimes it is useful to see just the first few, or last few, lines in a file. If you want to see what is in a file without looking at the whole file, you may find the **head** command useful. The **tail** command, which looks at the last few lines, can be useful as well. For example, you could use **tail** to look at the latest information in a log file, a file that is being constantly updated by some program on the system.

To look at the first few lines of a file, type **head *filename***, where ***filename*** is the name of the file you want to look at, and then press ⟨Enter⟩. By default, **head** shows you the first 10 lines of a file. You can change this by typing **head -*number filename***, where ***number*** is the number of lines you want to see.

For example, if you want to see the first 15 lines of */etc/passwd*, you could type:

head -15 /etc/passwd

To look at the last few lines of a file, use the **tail** command. **tail** works the same way as **head**: type **tail** and the filename to see the last 10 lines of that file, or type **tail** ***-number filename*** to see the last ***number*** lines of the file.

Try using **tail** to look at the last five lines of your *.profile* or *.login*:

1. Type **tail -5 $HOME/.profile** and press ⟨Enter⟩.
   (C shell users: type **tail -5 $HOME/.login** and press ⟨Enter⟩.)
2. **tail** displays the last five lines of your *.profile* (or *.login*).

```
% tail -5 $HOME/.login
        setenv  TERMCAP  $term[2]    # terminal data base
endif
unset term noglob
setenv PRINTER tpubs_lw1
setenv WPVER lyrix6                  # default lyrix version
```

Making directories

The **mkdir** command makes a directory on a UNIX system. To make a directory, change to the directory under which you want the new directory to live. Then, type **mkdir** ***directory***, where ***directory*** is the name you want to call the new directory, and press ⟨Enter⟩. If you have permission to write in the current directory, and there is no directory already named ***directory*** in the current directory, the new directory is created.

The rules on naming directories are the same as the rules on naming files: do not use a name longer than 256 characters, and do not use the filename metacharacters *, ?, [...]; otherwise, anything goes. A useful convention is to always start directory names with a capital letter. This way, you can differentiate between a file and a directory without doing a long listing (**l**).

You can make several directories at once by typing:

mkdir ***directory1 directory2 directory3***

Try creating a directory for memos and a directory for reports:

1. Type **cd** and press ⟨Enter⟩ to go to your home directory.
2. List the current directory (your home directory) by typing **lf** and pressing ⟨Enter⟩. Check that there are no files or directories named *Memos* or *Reports*.
3. Type **mkdir Memos Reports** and press ⟨Enter⟩. This creates two new directories, *Memos* and *Reports*.
4. Type **lf** and press ⟨Enter⟩ to list your home directory again. The **lf** command shows the two new directories.

```
$ cd
$ lf
$ mkdir Memos Reports
$ lf
Memos/    Reports/
```

Removing directories

You can remove directories using the **rmdir** command. To remove a directory, type **rmdir** and the name of the directory you want to remove.

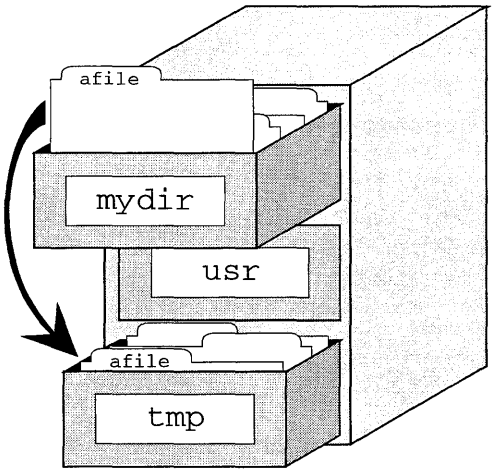
rmdir only removes directories that are empty. You can remove directories, subdirectories, and files all at once using options to the **rm** command, which is discussed later in this chapter. See **rm**(C) for more information.

Try creating a directory and then removing it:

1. Type **cd** and press ⟨Enter⟩ to go to your home directory.
2. Create a directory called *Letters*. (Type **mkdir Letters** and press ⟨Enter⟩.)
3. List the contents of the working directory (type **lf** and press ⟨Enter⟩), to confirm that the *Letters* directory has been created.
4. Remove the *Letters* directory by typing **rmdir Letters**, then pressing ⟨Enter⟩.
5. List the directory (type **lf** and press ⟨Enter⟩) to confirm that *Letters* has disappeared.

Copying files

The **cp** command copies files. To copy a file, type **cp**, the name of the file you want to copy, and the name you want to call the copy, then press ⟨Enter⟩. Unlike DOS, a UNIX system does not tell you that the copy succeeded, but it shows you an error message if it did not. You can use a pathname (a directory) for the name of the copy to put a copy of a file in a particular directory. In this case, unless you specify otherwise, the copied file will be given the same name as the original but in a different directory.



cp mydir/afile /tmp

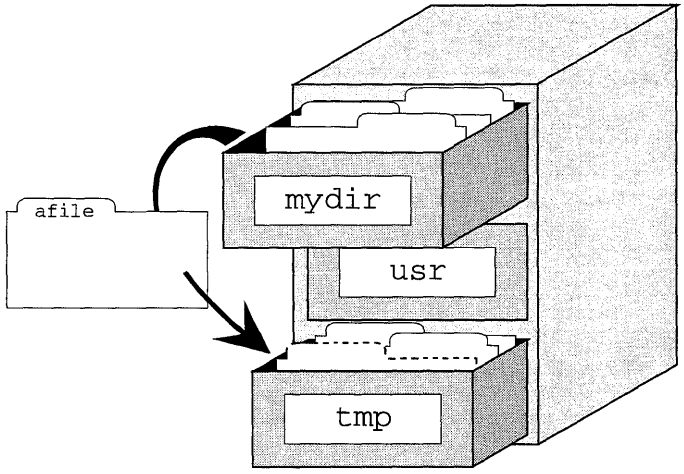
Try putting a copy of the message of the day file in your home directory:

1. Type **cp /etc/motd $HOME** and press ⟨Enter⟩.
2. Check if the copy was successful by typing **l $HOME/motd** and pressing ⟨Enter⟩. The computer shows you a copy of *motd* in your home directory if the copy worked.

```
$ cp /etc/motd $HOME
$ l $HOME/motd
-rw-r--r--  1 susannah techpubs   464 Jun 24 17:47 /u/susannah/motd
```

Renaming files

To rename a file on a UNIX system, use the **mv** (move) command. You can also use **mv** to "move" a file from one directory to another. To rename a file using **mv**, type **mv**, the name of the file you want to rename, the new name you want to call it, and press ⟨Enter⟩.



mv mydir/afile /tmp

For example, if you want to rename the file *mytodo* as *monday*, type:

mv mytodo monday

If you want to move *monday* to the temporary directory */tmp*, type:

mv monday /tmp

Try making a directory for to-do lists and moving the file *mytodo* into it. (If you do not have the file *mytodo* from the writing and editing lessons in Chapter 4, "Writing and editing" (page 41), use the **cat** command or the **vi** editor to create a file called *mytodo* containing a to-do list. See Chapter 4, "Writing and editing" (page 41) for instructions.)

1. If you are not already in your home directory, type **cd** and press ⟨Enter⟩.
2. Type **mkdir Todos** and press ⟨Enter⟩ to create a directory called *Todos*.
3. Move *mytodo* to the new directory by typing **mv mytodo Todos** and pressing ⟨Enter⟩.

4. Check the contents of the *Todos* directory by typing **ls Todos** and pressing ⟨Enter⟩.

```
$ mkdir Todos
$ mv mytodo Todos
$ ls Todos
mytodo
```

Removing files

Removing files you no longer need is an important part of managing the way you work on the computer. Any computer has a limited amount of disk space, and, although the computer may have a very large hard disk, eventually the disk begins to fill. To do your part in not adding to computer "litter," you should regularly remove files you no longer need.

To remove a file, type **rm**, the name of the file, and press ⟨Enter⟩. You can quickly remove a directory and all the subdirectories below it with the command **rm -rf** *. This command recursively removes everything in its path, asking no questions as it goes. **Be careful with it.**

Try creating a file and then removing it:

1. Type **cd** and press ⟨Enter⟩ to go to your home directory.
2. Use the **touch** command to create a file. Type **touch newfile** and press ⟨Enter⟩ to create a file 0 bytes long.
3. List the file by typing **l newfile** and pressing ⟨Enter⟩.
4. Remove *newfile* by typing **rm newfile** and pressing ⟨Enter⟩.
5. Check that you removed the file by typing **l newfile** and pressing ⟨Enter⟩. The computer responds with `newfile not found`.

```
$ cd
$ touch newfile
$ l newfile
-rw-r--r--  1 susannah techpubs     0 Jul 12 13:59 newfile
$ rm newfile
$ l newfile
newfile not found
```

Q: What if I see a message like `file: 600 mode`?

A: If you try to remove a file on which you do not have write permission, **rm** prints the filename followed by the permission mode of the file. This is the **rm** command's way of asking you if you are sure you want to remove the file. Type **y** to remove the file, or **n** to leave it as it is. For more information about permissions, see Chapter 7, "Protecting files and directories" (page 69).

Summary

To see a file screen by screen	**more** ***filename***
To see the first few lines of a file	**head** ***filename***
To see the last few lines of a file	**tail** ***filename***
To make a directory	**mkdir** ***dirname***
To remove an empty directory	**rmdir** ***dirname***
To copy a file	**cp** ***filename another_filename***
To rename a file	**mv** ***filename new_filename***
To remove a file	**rm** ***filename***

For more information about	See
Looking at files screen by screen	**more**(C)
Looking at the beginning of a file	**head**(C)
Looking at the end of a file	**tail**(C)
Making directories	**mkdir**(C)
Removing directories	**rmdir**(C)
Copying files	**cp**(C)
Renaming files	**mv**(C)
Removing files	**rm**(C)

Chapter 6

Commands revisited: pipes and redirection

In this chapter, you will learn how to put the results of a command into a file, how to use a file as input to a command, and how to put commands together to form customized utilities. You will also learn how to join files together, how to write information onto the end of a file, and how to put commands in the background to use the computer to do more than one task at once.

Before you begin, you should know what files and directories are, and how to create a file using **cat** or **vi**.

Putting the output of a command into a file

You have already seen one example of how to put the output of a command into a file: **cat > file**

Here, **cat** opens a file and waits for you to type into it. The file is closed when you press ⟨Ctrl⟩**D**, the end-of-file (EOF) character. The greater-than sign is the redirection symbol; it tells the computer you want the output of **cat** to go into a file instead of the usual place. This is called "redirecting standard output," or simply "redirection."

You can use redirection with any command that prints information on the screen. For example, you could redirect the output of **ls** into a file and then print this file to get a printed directory listing.

Try printing a long listing of the files in your home directory:

1. If you are not already in your home directory, go there by typing **cd** and pressing ⟨Enter⟩.
2. List the files into a file called *filelist*: type **l > filelist** and press ⟨Enter⟩.
3. Send the file *filelist* to the printer by typing **lp filelist** and pressing ⟨Enter⟩.

The usual place the output of a command goes is known as standard output. Standard output is usually your screen.

The usual place a command gets its input from is known as standard input. Standard input is usually your keyboard. You will learn how to use a file as standard input later in this chapter.

Standard output and standard input are sometimes referred to as "standard out" and "standard in," or "stdout" and "stdin."

Using a file as input to a command

Just as you can redirect the output of a command, you can redirect the input of a command. To tell a command to take input from a file, you type the command, then a less-than sign (<), then the file that you want it to use as input. The file used for input is still there after the command is finished; it is only read, it is not overwritten.

Suppose, for example, you wanted to mail a file called *report* to Doug you could type **mail doug < report**

This tells the **mail** command to take its input from *report*. This is a very fast way of mailing things because you never enter the interactive **mail** program, you just send the file.

Try mailing yourself a copy of */usr/adm/messages*, the file that stores system startup messages, using input redirection:

1. Type **mail *loginname* < /usr/adm/messages** and press ⟨Enter⟩. (Substitute your own login name for ***loginname***.)
2. Confirm that the file was sent by typing **mail** and pressing ⟨Enter⟩. Read your current messages; one of these contains the startup messages file.
3. Type **q** to quit **mail**.

Q: If I use redirection to mail a file to someone without entering the **mail** program, is there any way I can get a subject header on the message?

A: You can get a subject header on the file by using the **-s "*subject*"** option to **mail**. For example, to mail the file *prognotes* to Anne, you could type:

mail -s "Program notes" anne < prognotes

This sends the file with the subject heading "Program notes." For more information about **mail** options, see **mail**(C).

Joining files together

You can use the **cat** command to join files together without using an editor. To do this, type **cat**, the names of the files you want to join together, and then redirect the output into a new file. For example, if you want to join together *report1*, *report2*, and *report3* into a file called *allreps*, you could type:

cat report1 report2 report3 > allreps

cat opens a file called *allreps* and then writes each file, in order, into it. *report1* comes first in *allreps*, followed by *report2* and then *report3*.

Be careful with **cat**, because you can unintentionally overwrite a file. For example, type:

cat report1 report2 report3 > report1

cat first opens a file called *report1*, where it writes its input files. This overwrites the existing *report1*. When **cat** goes to write its arguments into the file *report1* that it has just opened, it finds *report1* appears as input as well as output and gives you the error message:

```
cat: input/output files 'report1' identical
```

However, by now it is too late; the contents of *report1* have been overwritten.

Background processing

The ability to run commands in the background is one of the key benefits of the UNIX system. You can set any command line running in the background while you do something else at the prompt.

To set a command running in the background, type the command at the prompt as usual, but type & (ampersand) after it, before you press ⟨Enter⟩. This tells the UNIX system you want the command to run in the background, so it immediately returns your prompt.

For example, if you have a lot of files to join together, or if the files are large, you can put the command in the background:

cat bigfile1 bigfile2 bigfile3 > bigfile&

When you put a command in the background, the computer responds with a number that is the process ID of the command. See Chapter 5, "Controlling processes" in the *Operating System User's Guide* for information about processes and process IDs.

Appending one file to another

You can use **cat** with redirection to append a file to another file. You do this by using the append redirection symbol, " >> ". To append one file to the end of another, type **cat**, the file you want to append, then >>, then the file you want to append to, and press ⟨Enter⟩.

For example, to append a file called *report2* to the end of *report1*, type:

cat report2 >> report1

You can use the append symbol " >> " with any command that writes output. For example, you could append a directory listing to a file called *log* with:

ls >> log

Try working through the following **cat** tutorial:

1. If you are not already in your home directory, go there by typing **cd** and pressing ⟨Enter⟩.
2. Use **cat** to create three files: *report1*, *report2*, and *report3*.

   Type **cat > report1** and press ⟨Enter⟩. Type **report 1** and press ⟨Enter⟩. Then, type **Keeping a cat is a serious responsibility** and press ⟨Enter⟩. Now, press ⟨Ctrl⟩**D**.

Create *report2* by typing **cat > report2** and pressing ⟨Enter⟩. Type **report 2** and press ⟨Enter⟩. Then, type **Cats need a balanced diet** and press ⟨Enter⟩. Now, press ⟨Ctrl⟩**D**.

Create *report3* by typing **cat > report3** and pressing ⟨Enter⟩. Type **report 3** and press ⟨Enter⟩. Then, type **Responsible cat owners will neuter or spay their pets** and press ⟨Enter⟩. Now, press ⟨Ctrl⟩**D**.

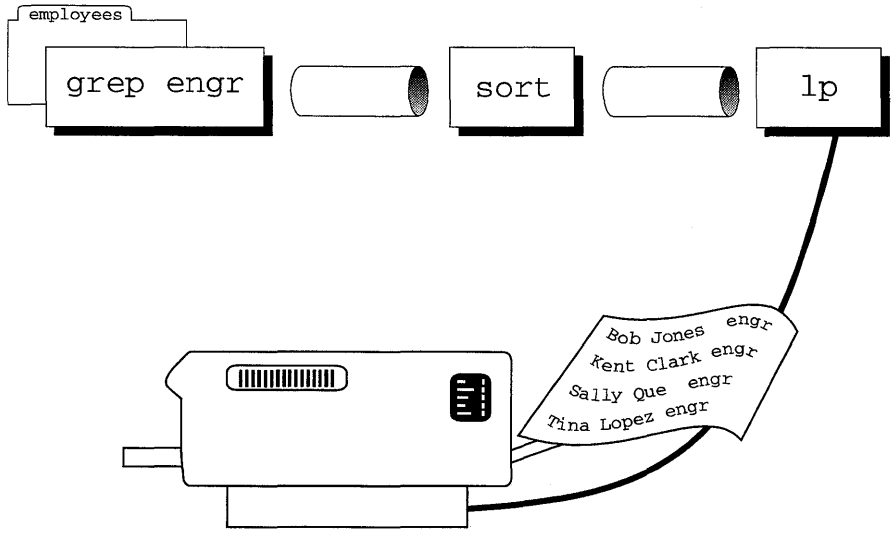
```
$ cat > report1
report 1
Keeping a cat is a serious responsibility
⟨Ctrl⟩D
$ cat > report2
report 2
Cats need a balanced diet
⟨Ctrl⟩D
$ cat > report3
report 3
Responsible cat owners will neuter or spay their pets
⟨Ctrl⟩D
```

3. Now, put the three files together into a file called *allreps*. Type **cat report1 report2 report3 > allreps** and press ⟨Enter⟩.
4. Look at *allreps* by typing **cat allreps** and pressing ⟨Enter⟩.
5. Now try using the append symbol to put the files together in the reverse order. Type **cat report3 report2 > repsagain** and press ⟨Enter⟩. Then type **cat report1 >> repsagain**.
6. Type **cat repsagain** and press ⟨Enter⟩ to see what happened.

```
$ cat report1 report2 report3 > allreps
cat allreps
report 1
Keeping a cat is a serious responsibility
report 2
Cats need a balanced diet
report 3
Responsible cat owners will neuter or spay their pets
$ cat report3 report2 > repsagain
$ cat report1 >> repsagain
$ cat repsagain
report 3
Responsible cat owners will neuter or spay their pets
report 2
Cats need a balanced diet
report 1
Keeping a cat is a serious responsibility
```

Using pipes to build your own utilities

You can use the pipe symbol (|) on a UNIX system to make the output of one command the input of another. To do this, you type the command you want to generate the input, a pipe symbol, the command you want to read the input, and then press ⟨Enter⟩. You can use pipes to put together as many commands as you like.



grep engr employees | sort | lp

Earlier in this chapter, you learned how to print a directory listing by typing:

l > filelist
lp filelist

Doing this with a pipe is even faster:

l filelist | lp

Another way of using a pipe is to put long output through the **more** command.

l /etc | more

Try using a pipeline to print a list of the files in your home directory:

1. If you are not already in your home directory, go there by typing **cd** and pressing ⟨Enter⟩.
2. Type **l | lp** and press ⟨Enter⟩ to send a long listing straight to the printer.

Summary

To put the output of a command into a file	***command_line*** **>** ***filename***
To use a file as input to a command	***command_line*** **<** ***filename***
To join files together	**cat** ***file1 file2 file3*** **>** ***newfile***
To append one file to another	**cat** ***file*** **>>** ***logfile***
To send a file listing to the printer	**l | lp**

For more information about	See
All the topics covered in this chapter	Chapter 3, "Working with files and directories" in the *Operating System User's Guide* **csh**(C), **ksh**(C), and **sh**(C)

Chapter 7

Protecting files and directories

In this chapter, you will learn about user identification, group identification, and permissions that the UNIX system uses to keep files secure. You will learn how to read the information in a long listing, and how to change the owner, the group, and the permissions of a file.

Before you begin, you should be familiar with files and directories. You should know how to use the **l** command to get a long listing and you should know how to use **vi** to edit a file.

Reading a long listing

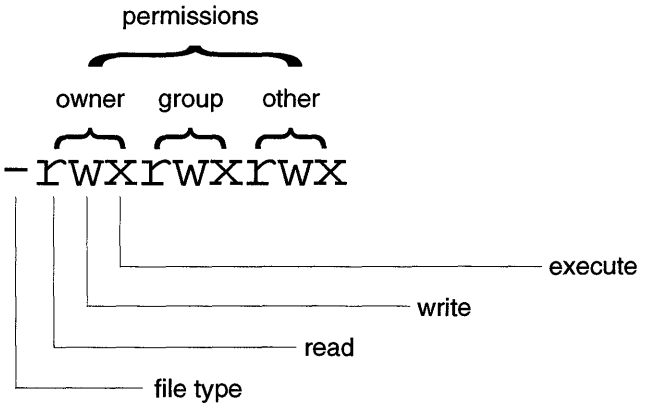
In "Listing more information about files" (page 37), you learned how to use the **l** command to create a long listing (remember to use the **-L** option to see the files rather than the links):

```
$ ls -l -L /etc
total 4566
-rwx------   1 bin      bin        10510 Mar 16  1993 aiolkinit
-rw-rw-r--   1 root     mem           48 Jun 17 08:34 arp_data
-rwx------   2 bin      bin         2966 Mar 15  1993 asktime
-rwx------   2 bin      bin         2966 Mar 15  1993 asktimerc
drwxrwx--x   4 auth     auth          80 Mar 24  1993 auth
-r-x------   1 bin      auth        1706 Mar 15  1993 authckrc
-rwx--x--x   1 root     root      128266 Mar 12  1993 automount
-rw-r--r--   1 root     root           0 Jun 17 08:34 advtab
...
```

In a long listing you see the permissions, the number of links, the owner, the group, the size in bytes, the modification date, and the name of the file. In this section, we will be looking at the permissions, the owner, and the group.

Permissions

The UNIX operating system stores a set of permissions with every file. These permissions help to keep files secure on a multiuser system by determining who can access a file or a directory, who can change a file, and who can run a program.



A file's permissions are shown in the first field of a long (**l**) listing. The permissions field is made up of 10 places; think of this as 1 place + 3 places + 3 places + 3 places. For example:

```
-r--r--r--   1 root   techpubs   3026 Jul 03 09:40 /etc/passwd
```

Each place can contain a character indicating a particular permission. The most common permissions are:

Permission	Meaning	Where it can occur
d	Directory permission	First place, before the 3 sets of 3
r	Read permission	First place in each set of 3
w	Write permission	Second place in each set of 3
x	Execute permission	Third place in each set of 3

If a place in a permission listing contains a hyphen (-) instead of a permission character, it means that permission (read, write, or execute) does not apply to that set of three.

Read permission lets you read a file, copy a file, print a file, change into a directory, and so on.

Write permission lets you modify a file, create a file in a directory, and remove a file from a directory. (To remove a file with **rm**, you only need write permission in the file's directory. You can then remove files on which you do not have write permission, although **rm** will prompt you for confirmation before it removes them.) Execute permission lets you run a compiled program or a shell script. (A shell script is a text file of shell programming commands and regular UNIX system commands that the shell executes one line at a time. For more information about shell scripts, see Chapter 9, "Customizing your environment".)

Directory permission is not really a permission at all; it simply indicates a file that is a directory. (Internally, the UNIX system stores files and directories the same way; it thinks of a directory as a special kind of file.)

Owner, group, other

The permissions field of a file is divided into 1 + 3 + 3 + 3 places to allow you to set different permissions for different users. The first place (1) is reserved for file types such as "d" for directory and "-" for regular files. This is not user-specific. Each of the following sets of three applies to a particular set of users.

The first set of three permissions, after the place for **d**, applies to the *owner* of the file, the user whose name appears in the third field of the long listing.

The second set of three permissions applies to the all users who are members of the *group* of the file. (The group of a file is shown in the fourth field of a long listing.)

The third set of three permissions applies to *others*; that is, to everyone who is not the owner of the file, and is not a member of the group of a file.

These three sets of three permissions are known as owner, group, and other.

Q: What is a group?

A: Just as every person who uses the computer has a login name, every person is also a member of a group. Groups, together with group permissions, allow people who need to use the same files to share those files without sharing them with all other users.

For example, if you wrote a report that you wanted members of your department to read, but not everyone else in the company, you could set permissions like:

```
-rw-r-----   1 susannah techpubs      25 Jun 27 11:58 report
```

This would allow you to modify the file (the r and the w in the first set of three), allow other members of your group (here, *techpubs*) to read the file (the r in the second set of three), and prevent others from reading or modifying the file (the three hyphens in the third set of three). The first place contains a hyphen because *report* is a file, not a directory.

Q: How can I tell what group I am in?

A: The **id**(C) command shows you your login name and your group. Type **id** and press ⟨Enter⟩. You see something like:

```
$ id
uid=12846(susannah) gid=1014(techpubs)
```

The computer is showing you your login name and group information in the way it thinks of them: as a UID (user identification), and a GID (group identification). The UID is a numeric representation of your login name; the GID is a numeric representation of your group. **id** shows the login name and group name in parentheses following the UID and GID.

Changing the group of a file

You can change the group of a file using the **chgrp** command if you are the owner of that file. If you need to change the group of a file that you do not own, you must ask the owner of the file to do this. You can also ask your system administrator, who can use the superuser (*root*) account to modify any file.

To change the group of a file, type **chgrp**, the name of the new group, the name of the file, and press ⟨Enter⟩. For example, to change the group of the file *report* to a group called *unixdoc*, you could type:

```
$ l report
-rw-r-----   1 susannah techpubs      25 Jun 27 11:58 report
$ chgrp unixdoc report
$ l report
-rw-r-----   1 susannah unixdoc       25 Jun 27 11:58 report
```

Changing the owner of a file

You can use the **chown** (change owner) command to change the ownership of a file that you own. As with **chgrp**, only the owner of a file or the superuser (*root*) can change the ownership of that file.

To use **chown**, type **chown**, the login name of new owner of the file, the name of the file you want to change, then press ⟨Enter⟩.

For example, to change the owner of the file *report* from the previous example, you could type:

```
$ l report
-rw-r-----   1 susannah unixdoc       25 Jun 27 11:58 report
$ chown root report
$ l report
-rw-r-----   1 root     unixdoc       25 Jun 27 11:58 report
```

Changing the permissions on a file

To change the permissions on a file, you use the command **chmod**. (**chmod** stands for "change mode;" a file's permissions are also known as its mode.) As with **chown**, and **chgrp**, only the owner of a file or the superuser (*root*) can change the permissions of a file.

To change the permissions on the file, type **chmod**, how you want to change the permissions, the name of the file, then press ⟨Enter⟩.

To specify how you want to change permissions, you type a letter representing which set of permissions you want to change, a symbol that tells whether you want to add to, remove from, or overwrite the existing permissions, and a letter representing which permission you want to work with.

For example, to change the permissions on the file *report* so that members of the group *techpubs* can modify the file, you could type:

```
$ l report
-rw-r-----   1 susannah techpubs      25 Jun 27 11:58 report
$ chmod g+w report
$ l report
-rw-rw----   1 susannah techpubs      25 Jun 27 11:58 report
```

The **chmod** command in the preceding example says "group plus write"; in other words, add write permission to the existing permissions for group. If you wanted to remove the group write permission, you could type:

```
$ l report
-rw-rw----   1 susannah techpubs      25 Jun 27 11:58 report
$ chmod g-w report
$ l report
-rw-r-----   1 susannah techpubs      25 Jun 27 11:58 report
```

If you wanted to remove all permissions for group, you could type:

```
$ l report
-rw-rw----   1 susannah techpubs      25 Jun 27 11:58 report
$ chmod g= report
$ l report
-rw-------   1 susannah techpubs      25 Jun 27 11:58 report
```

The equals sign in the second example says "overwrite all group permissions with nothing"; in other words, remove all group permissions.

You can think of how you specify permissions as an expression of the form:

chmod ***who*** [+ | - | =] ***permission filename***

Here, ***who*** tells which set of permissions you want to change; + , -, or = tells whether you want to add, remove, or overwrite; ***permission*** is the permission itself, and ***filename*** is the name of the file.

Here are all the options for ***who***:

Option	Meaning
a	All users; change all three sets of permissions at once
u	User; change the user, or owner, permissions
g	Group; change the group permissions
o	Others; change the other permissions

If you do not specify a ***who*** (for example, if you just said **chmod +w**), the write permissions are changed for all three sets.

Try creating a *report* file and then changing the permissions, the ownership, and the group it is in:

1. If you are not already in you home directory, type **cd** and press ⟨Enter⟩. (The reason you type **cd** and press ⟨Enter⟩ at the beginning of each exercise is to guarantee you are working in a directory where you have write permission.)

2. Create a file called *test* using **cat**. Type **cat > test** and press ⟨Enter⟩; then type **This is a test file** (or whatever text you like); then press ⟨Enter⟩ to go to a new line and press ⟨Ctrl⟩**D** to close the file.

```
$ cd
$ cat > test
This is a test file
⟨Ctrl⟩D
```

3. List the file by typing **l test** and pressing ⟨Enter⟩.

4. Change the permissions on the file so that everyone can modify the file by typing **chmod +rw test** and pressing ⟨Enter⟩.

5. Check what happened by typing **l test** and pressing ⟨Enter⟩.

6. Now, change the permissions back so only the owner of the file can change it, by typing **chmod o-w test** and pressing ⟨Enter⟩, **chmod g-w test** and pressing ⟨Enter⟩.

7. List the file with **l test** and press ⟨Enter⟩.

8. Change the ownership of the file to *root* by typing **chown root test** and pressing ⟨Enter⟩; then list the file with **l test** and press ⟨Enter⟩.

```
$ l test
-rw-rw----   1 susannah techpubs      20 Jun 27 15:40 test
$ chmod +rw test
$ l test
-rw-rw-rw-   1 susannah techpubs      20 Jun 27 15:40 test
$ chmod o-w test
$ chmod g-w test
$ l test
-rw-r--r--   1 susannah techpubs      20 Jun 27 15:40 test
$ chown root test
$ l test
-rw-r--r--   1 root     techpubs      20 Jun 27 15:40 test
```

9. Use the **vi** editor to open the file (**vi test** and ⟨Enter⟩). You should see the words `"test" [Read only]` at the bottom of your screen. This is because you are no longer the owner of the file, so you only have read permission on it. (If you were to make changes to the file, when you tried to save it, you would see the error message `File is read only`; you would not be able to save your changes.) Type **:q** to quit **vi**.

Summary

To change the group of a file	**chgrp** ***newgroupname filename***
To change the owner of a file	**chown** ***newowner filename***
To change the permissions on a file	**chmod [u | g | o] [+ | - | =] [r | w | x]** ***filename***

For more information about	See
Long file listings	**ls**(C)
Changing the owner of a file	**chown**(C)
Changing the group of a file	**chgrp**(C)
Changing the permissions on a file	**chmod**(C)

Chapter 8

Power tools

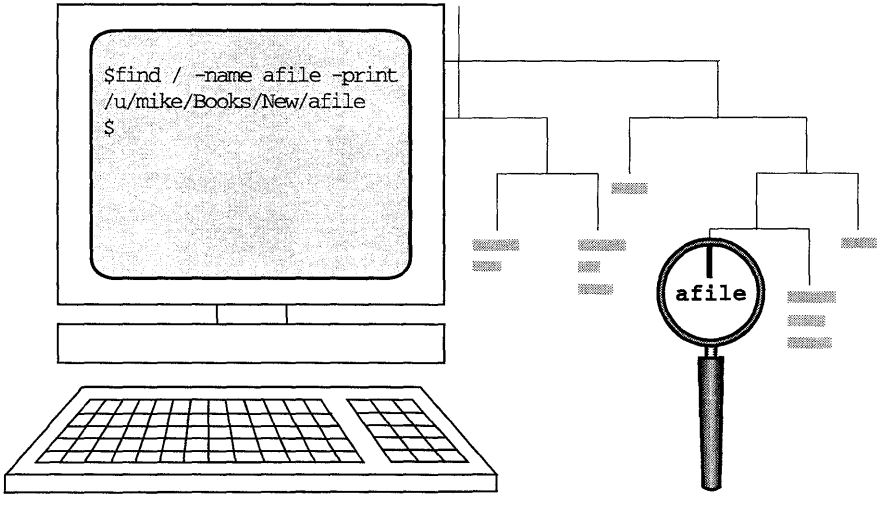
In this chapter, you will learn the basics of two of the most powerful UNIX system tools: **grep** and **find**. You will also be introduced to a handful of other UNIX utilities that may make working on the computer more productive, more fun, or both.

Before you begin, you should feel comfortable with what you have learned so far. You should know how to move from directory to directory, how to list files, and how to read a long file listing. You should know how to use **mail**, **cat**, and **vi**. You should know how to manage your files, and how to modify their permissions.

Most importantly, you should be learning how to combine commands together with pipes, and how to use redirection to take input from, or put output into, a file. The commands you will learn in this chapter are powerful on their own, but they can be much more powerful when combined with some of the commands you have already learned.

Searching for a file

You can use the **find** command to find a file anywhere on the system.



find / -name afile -print

To use this command, type **find**, the name of the directory where you want it to start looking, **-name** and the name of the file you want to find, **-print**, then press ⟨Enter⟩. For example, to look for a file called *rts* starting in the directory */etc*, you could type:

```
$ find /etc -name rts -print
/etc/perms/rts
```

The **find** command in the preceding example says, "Find a file named *rts*, and print the pathname when you find it. Start looking in the directory */etc*."

The **find** command starts from the directory you specify and looks through every directory below it for files with names that match the file you put after **-name**. If **find** runs across directories where you do not have read permission, it gives you an error message like `find: cannot open` ***directoryname***. If you are trying to find something starting from the *root* directory (/), which could take some time, you may want to redirect the output to a file and put the whole task in the background:

find / -name mytodo -print > foundfile &

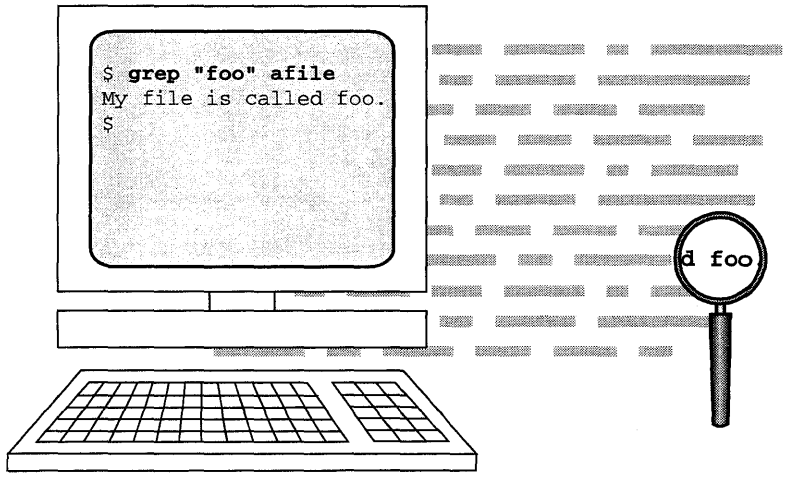
The **find** command has many options in addition to what you have seen here.

Using **find**, you can locate a file based on any of its attributes, for example, its owner, its size, or the time it was last modified. You can also tell **find** to perform a particular command when it finds a file; for example, you can have **find** look for all files older than a certain date and remove them.

Because **find** produces output containing absolute pathnames of everything it found, **find** can be a useful beginning of a pipeline anytime you need to generate a lot of pathnames. For example, system administrators who use the **cpio**(C) backup program use **find** to generate a list of files to be backed up, and then they pipe the **find** output through **cpio** to do the actual backing up. See **find**(C) for information about all the **find** options.

Searching for text within files

You can use the **grep** command to search through files for a particular pattern.



grep "foo" afile

The name **grep** comes from the **ed**(C) command **g/re/p**, which means "globally search for a regular expression and print it". The **grep** command looks through the files you specify for lines containing the regular expression you tell it to find.

Regular expressions are a special kind of pattern that are used by many UNIX system commands. You can use regular expressions with **vi**, as well as with **ed**(C), a line editor; **sed**(C), a stream editor; **expr**(C), an expression evaluator; and **awk**(C), a regular-expression-based programming language.

A discussion of regular expression syntax is beyond the scope of this book; see **ed**(C) for all the details.

To use **grep** to search for words in a file, type **grep**, the word or words you want to search for, the files you want to look in, and press ⟨Enter⟩. If you want to look for more than one word, you need to put "double quotes" around the words.

If you specify more than one file for **grep** to look in, **grep** tells you the name of the file in which the word was found, and it shows you the line in which the word appears. If you only specify one file to look in, **grep** does not tell you the name of the file. If **grep** cannot find the word in the specified file or files, it silently returns your prompt.

For example, to search for an entry in */etc/passwd*, you could type:

```
$ grep susannah /etc/passwd
susannah:IHi3E6JQ.IJxU:12846:1014:Susannah Skyer:/u/susannah:/bin/sh
```

To look through all the files in the current directory for the words "cat food," you could type:

```
$ grep "cat food" *
mytodo:buy cat food
```

In the preceding example, "cat food" was found in the file *mytodo*, in the line "buy cat food."

You can use **grep** together with other commands to search for particular lines of output. For example, to see all of the files owned by *susannah* in */tmp*, you could type:

```
$ l /tmp | grep susannah
-rw-------   1 susannah unixdoc        0 Jun 24 16:29 Ex05064
-rw-------   1 susannah techpubs    8192 Jun 27 16:57 Ex29109
-rw-rw----   1 susannah techpubs    3532 Jun 24 15:48 maila14986
-rw-------   1 susannah techpubs    2048 Jun 27 16:55 Rx29109
-rw-rw----   1 susannah techpubs    4960 Jun 24 13:32 unixmeet
```

There are a variety of options with the **grep** command, and there are also two faster versions of **grep**, **fgrep** (fast **grep**) and **egrep** (expression **grep**), which you can use in some instances. See **grep**(C) for more information.

Checking who is logged in

You can use the **who** command to find out who is logged in, where they are logged in, and when they logged in. To use **who**, simply type **who** and press ⟨Enter⟩:

```
$ who
backup     tty01       Jun 28 07:56
perry      tty002      Jun 28 09:14
theresma   tty003      Jun 28 08:49
joseph     tty004      Jun 28 11:20
liz        tty005      Jun 28 09:02
cecile     tty006      Jun 28 10:06
kate       tty007      Jun 28 09:35
liane      tty009      Jun 28 09:41
bridget    tty011      Jun 28 08:06
sarah      tty001      Jun 28 08:06
nigel      tty008      Jun 28 08:02
gudrun     tty010      Jun 28 08:09
susannah   tty013      Jun 28 09:14
```

The tty number that follows each person's login name tells which terminal they logged in on. If people are logged in on several terminals at once, they appear once per login in the **who** listing.

You can find out when someone logged in by searching for their login name using **grep**:

```
$ who | grep sarah
sarah      tty001      Jun 28 08:06
```

Finding out more information about a user

The **finger** command shows you more information about a user. To use **finger**, type **finger**, the login name of the person you want to find out more information about, and press ⟨Enter⟩.

For example:

```
$ finger sarah
Login name: sarah                       In real life: Sarah Connell
Directory: /u/sarah                     Shell: /usr/sco/bin/ksh
On since Jun 28 08:06:50 on tty001       14 minutes Idle Time
No Plan
```

The information **finger** shows you depends on how it has been set up on your system. If you create a file called *.plan* in your home directory, the information in this file is shown when someone types **finger** and your login name:

```
$ cd $HOME
$ cat > .plan
To finish writing the tutorial and go on a long, long vacation
⟨Ctrl⟩D
$ finger susannah
Login name: susannah                    In real life: Susannah Skyer
Directory: /u/susannah                  Shell: /bin/sh
Not logged in.
Plan:
To finish writing the tutorial and go on a long, long vacation
```

Finding out the time and date

The **date** command shows you the current time and date:

```
$ date
Mon Jun 27 14:19:48 BST 1994
```

Seeing a calendar

You can see a calendar using the **cal** command. If you type **cal** and press ⟨Enter⟩, you see a calendar for last month, this month, and next month, along with the current time and date.

```
$ cal
Thu Jun 23 16:25:09 1994
          May                    Jun                    Jul
 S  M Tu  W Th  F  S    S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
 1  2  3  4  5  6  7                1  2  3  4                    1  2
 8  9 10 11 12 13 14    5  6  7  8  9 10 11    3  4  5  6  7  8  9
15 16 17 18 19 20 21   12 13 14 15 16 17 18   10 11 12 13 14 15 16
22 23 24 25 26 27 28   19 20 21 22 23 24 25   17 18 19 20 21 22 23
29 30 31               26 27 28 29 30         24 25 26 27 28 29 30
                                              31
```

You can see a calendar for a particular month and year by typing **cal**, then the month (or an abbreviation of it), then the year. For example, to see the calendar for August 1996, you could type:

```
$ cal Aug 1996
   August 1996
 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 31
```

You must type the year in full; if you type **cal Sept 96**, you see the calendar for September 96, not September 1996.

Remembering your appointments

You can use the **calendar** reminder service to send yourself mail reminders about upcoming appointments. The **calendar** program looks for a file called *calendar* in your home directory and then mails you the lines containing today's or tomorrow's date.

To use **calendar**, create a file called *calendar* in your home directory. Put each appointment on a single line containing the date of the appointment in US (month-day) format. For example:

```
$ cd
$ cat > calendar
7/4 Dave and Peter start work
7/5 UNIX Beta Committee meeting, 10 am
7/7 Meet Vip at the airport
⟨Ctrl⟩D
```

Then, to guarantee that the **calendar** program reads your *calendar* file each time you log in, add the following line to the end of your *.profile* (C shell users: add this line to your *.login*):

```
/usr/bin/calendar
```

For more information on customizing your *.profile* or *.login* see "Your environment" (page 87) and Appendix D, "Sample shell startup files" in the *Operating System User's Guide.*

Using a calculator

You can use the **bc** program as an online calculator. (**bc** is actually a programming language, similar to C, which is used for performing mathematical calculations.) To use **bc**, type **bc** and press ⟨Enter⟩ — this brings you into **bc**. You can now type any arithmetic expression, press ⟨Enter⟩, and **bc** evaluates it for you. When you are done using **bc**, press ⟨Ctrl⟩D to return to the prompt.

For example, to use **bc** to add up your work hours, you could type:

```
$ bc
7+6.5+8+8.5+7
37.0
⟨Ctrl⟩D
```

Clearing the screen

The **clear** command clears your screen. Some people like to clear their screens when they begin working on a new task, so they can start with a "clean slate." You may also want to clear your screen if you step away from your desk for a moment, although it is a better practice to log out.

Summary

To search for a file	**find / -name *filename* -print**
To search for text within a file	**grep "*text* " *filename***
To see who is logged in	**who**
To find out more information about a user	**finger *username***
To find out the time and date	**date**
To see a calendar	**cal**
To be reminded of appointments	Create a *calendar* file in your home directory
To use a calculator	**bc** ***calculator commands*** ⟨Ctrl⟩**D**
To clear the screen	**clear**

For more information about	See
Searching for files	**find**(C)
Searching for text within files	**grep**(C)
Checking who is logged in	**who**(C)
Finding out more information about a user	**finger**(C)
Finding out the time and date	**date**(C)
Seeing a calendar	**cal**(C)
Using the calendar reminder service	**calendar**(C)
Using a calculator	**bc**(C)
Clearing the screen	**clear**(C)

Chapter 9

Customizing your environment

In this chapter, you will learn how to tailor the way you work on the UNIX system by editing the files the computer reads every time you log in.

Before you begin, you should have worked through the other chapters in this book. You should know which shell you are using, and you should be able to use **vi** to edit a file.

Your environment

The UNIX system uses the term "environment" to refer to all the settings that influence the way you work on the computer.

You can define the following sort of information in your environment:

- Your prompt.
- What directories are searched in what order when the computer looks for the commands you type.
- What permissions are assigned to the files you create.
- How often your shell looks for new mail.
- Where mail you have read is saved.
- What name you type to use a particular command.

How you set this information depends on which shell you are using.

Each shell has certain control files that it reads when you log in. For the Bourne shell (**sh**), the control file is called *.profile*. The Korn shell (**ksh**) has both a *.profile* and a *.kshrc*, and the C shell (**csh**) has a *.login* and a *.cshrc*.

The difference between *.profile* and *.kshrc*, and between *.login* and *.cshrc*, is in when the files are read. The *.profile* and the *.login* are only read when you log in. However, the *.rc* files, *.kshrc* and *.cshrc*, are read each time you start a **ksh** or **csh**. (You can start a shell from the command line by typing the name of the shell just like you would type any command.)

These control files are shell scripts: "shell" because they are written in the shell programming language; "scripts" because they are text files that are read one line at a time, like a DOS batch file.

In shell scripts, you see commands you are already familiar with, as well as programming constructs for looping, branching, and setting variables.

For listings and explanations of a typical *.profile*, *.kshrc*, *.login*, and *.cshrc*, see Appendix D, "Sample shell startup files" in the *Operating System User's Guide*.

Changing your prompt

A prompt will appear after you have logged into your system. The UNIX system stores this prompt in a variable.

To change your prompt, you reset the value of the prompt variable.

In the Bourne and Korn shells, the prompt variable is called **PS1** (prompt string 1). In the C shell, the prompt variable is called **prompt**.

All three shells have a secondary prompt as well as the main prompt. This secondary prompt is shown when you type a command that makes the shell expect further input.

For example, in the Bourne shell, the secondary prompt is ">" by default:

```
$ for i in *.tut
>
```

Here, you are saying to the shell "for every file (**i** represents every file) ending in .tut ..." The shell gives you a secondary prompt because it needs more information to complete your command.

In the Bourne and Korn shells, the secondary prompt is stored in the variable **PS2** (prompt string 2), which you can reset. You cannot reset the secondary prompt in the C shell. To reset **PS2** in **sh** or **ksh**, follow the instructions below, substituting **PS2** for **PS1**.

To reset your prompt in the Bourne or Korn shells, type **PS1=*value*** where ***value*** is the value you want to assign to the prompt variable.

For example, to set your prompt to say "Yo", you would add the following line to your *.profile* or *.kshrc*:

```
PS1=Yo
```

C-shell users would add the following line to their *.login*:

```
set prompt=Yo
```

Q: When I change my prompt, I lose the space between my prompt and where the command line starts. How do I get this back?

A: To keep the space between the prompt and the command line, you need to put a space after your new prompt. To get the shell to notice the space, you need to enclose the whole prompt string in double quotes.

In **ksh** or **sh**, add the following line to your *.profile*:

```
PS1="Hello friend "
```

In **csh**, add the following line to your *.login*:

```
set prompt="Hello friend "
```

Q: How do I get my prompt to show the current directory, like on DOS?

A: In **sh**, add the following line to your *.profile*:

```
nd()    { cd $* ; PS1="`pwd` "; }
```

Now, use the command **nd**, which you just created, to change to a new directory and display the directory as the prompt.

In **ksh**, add the following line to your *.kshrc*:

```
PS1='$PWD '
```

In **csh**, add the following lines to your *.cshrc*: (You do not need to add the lines that start with #. These are comments.)

```
# make a command doprompt that sets the prompt to the working directory
alias doprompt 'set prompt="`pwd` "'
# set the prompt the first time around
doprompt
# alias the cd command to change directories and reset the prompt
alias cd 'chdir * || doprompt'
```

Setting your path

Each command you type is actually a program that is stored somewhere on the computer. When you type a command and press ⟨Enter⟩, your shell looks through all the directories in your path until it finds a program with the same name as the command you typed.

When you see a message like `"not found"`, it means your shell could not find the command in any of the directories listed in your path. If you see a `"not found"` message for a command that you know exists, ask your system administrator what directory the command lives in, then add that directory to your path definition. In the meantime, you can type the full pathname of the command, for example, **/usr/bin/finger**. When you use the full pathname of a command, you tell the shell exactly where the command lives, so it does not search through the directories in your path definition.

A typical path setting in a **sh** or **ksh** *.profile* might look like this:

```
PATH=/bin:/usr/bin:$HOME/bin:.
```

This says "set the path to look in the *bin* directory, then */usr/bin*, then the *bin* directory in the home directory, and finally, in the current directory."

The same path setting in **csh** *.login* would be:

```
set path=(/bin /usr/bin $HOME/bin .)
```

To add a directory to your path, simply edit the path statement in your *.profile*, *.login*, *.kshrc*, or *.cshrc* to contain the new directory. For example, to add the directory */usr/company/bin* to your path in **sh** or **ksh**, you could change your path statement to read:

```
PATH=/bin:/usr/bin:/usr/company/bin:$HOME/bin:.
```

Q: Why would I want to put a new directory in the middle of the path definition instead of at the end?

A: You control the order in which directories are searched by the order you put those directories in the path definition. In general, you want to put nonstandard directories, like your company *bin* and your personal *bin*, after the standard */bin* and */usr/bin*. This is because most of the commands you want to use are in these standard directories, so putting them at the beginning of your path means your shell finds them more quickly.

Q: My path setting contains the **PATH** variable itself:

```
PATH=$PATH:$HOME/bin
```

What does this mean?

A: A path setting like this says "set the path to the current path, then add in the *bin* in my home directory." When you log in, your shell first reads definitions from the system-wide profile */etc/profile*. If your system has been set up so that */etc/profile* contains path definitions, including **$PATH** in your path definition ensures that your shell knows about any system-wide path definitions.

Default file permissions

You have already seen how the UNIX system uses file and directory permissions to control who can access which files. So far, you have learned to manipulate these permissions using symbolic mode, like:

chmod a+x newfile

This says, "Change the mode of *newfile* so all users have execute permission." Before you learn how to control a file's default permissions, you need to understand how to specify permissions using absolute mode.

Changing permissions with absolute mode

When you specify permissions using absolute mode, you use a three-digit octal number to specify the permissions for owner, group, and other.

For example, if you wanted to change the permissions on a file so that the owner had read and write permission, members of the group had read permission, and no one else had any permissions, you could type:

chmod 640 *file*

Here ***file*** is the name of the file.

In the preceding example, 640 is an octal number representing file permissions. The 6 represents the permissions for owner, the 4 is the permissions for group, and the 0 is the permissions for other. These digits are arrived at by taking the binary value of each permission, read, write, or execute, and adding them together to form one octal digit that represents the whole set (owner, group, or other).

Here are the octal values for some common permission settings:

Permission	Value
r	4
w	2
x	1
r+w	6
r+x	5
all permissions	7

To change a file's permissions to `r--------`, you could type:

chmod 400 *file*

To change a file's permissions to `rwxrwxr-x`, you could type:

chmod 775 *file*

As you can see, once you are used to changing permissions using absolute mode, it can be a quicker method than symbolic mode.

Setting your file creation mask

To control the default permissions that are given to every new file you create, you use the umask (user mask) command.

The **umask** command sets up a file creation mask. Setting a mask is the opposite of setting the permissions themselves; when you set a mask, you are telling the computer the permissions you do not want, rather than the permissions you do.

To set the default file permissions on new files you create to `rw-r-----`, you could add the following line to your *.profile* or *.login*:

```
umask 137
```

This is the opposite of saying **chmod 640**. If you wanted to set a **umask** for `rw-rw----`, it would be:

```
umask 117
```

A **umask** that allowed read and write permission for everyone would be:

```
umask 111
```

A **umask** that denied permissions to everyone except the owner of the file would be:

```
umask 177
```

You can see your current **umask** by typing **umask** and pressing ⟨Enter⟩. If **umask** is not explicitly set in one of your shell startup files, the computer shows you a default **umask**.

You can change your **umask** at the command line by typing **umask**, then the value you want your mask to have, then pressing ⟨Enter⟩. Keep changing your **umask** and creating and listing files until you get the default permissions you want.

Configuring mail

In this section you will learn how to control where your shell looks for mail and when and how it tells you that new mail has arrived. These are options you can control within your shell startup files. You can also set a variety of mail options in **mail**'s own startup file, *.mailrc*. For information about the options you can set in *.mailrc*, see **mail**(C).

Depending on the shell you are using, you can specify where mail is looked for by setting the **MAILPATH** or the **MAIL** variable. Again, depending on your shell, you can control how often you are notified of new mail by setting the **MAILCHECK** or **MAIL** variable.

To tell your shell where to look for mail, set the appropriate variable to the pathname where you receive your mail. (Generally, you receive mail in */usr/spool/mail/**loginname***, where ***loginname*** is your login name. If you are unsure about where you receive your mail, ask your system administrator.) With **sh** and **ksh**, you can tell your shell how you want to be prompted for new mail using this same variable setting.

To set your mail path and new mail message in **sh**, add the following line to your *.profile*:

```
MAILPATH=pathname%message
```

Here ***pathname*** is the pathname and ***message*** is the message you want to be prompted with. For example:

```
MAILPATH=/usr/spool/mail/susannah%Yo, you've got some new mail
```

To set your mail path in **ksh**, add the following line to your *.profile* or *.kshrc*:

```
MAILPATH=pathname?message
```

This is the same as with the Bourne shell, only you use a ? to introduce the message you want to see. If you leave out the message, **sh** prints `you have new mail` and **ksh** prints `You have new mail`.

To set your mail path in **csh**, add the following line to your *.login* or *.cshrc*:

```
set MAILPATH=pathname
```

Here ***pathname*** is the pathname from where you want your mail read. **csh** prompts you with `You have new mail` when new mail arrives; you cannot change this.

By default, each shell checks for mail every 10 minutes. You can change this by specifying a new time in seconds. In **sh** or **ksh**, add the following line to your *.profile* or *.kshrc*:

```
MAILCHECK=seconds
```

Here ***seconds*** is the number of seconds you want to go by before the shell checks for new mail again. For example, if you want your **ksh** or **sh** shell to check for mail every half hour:

```
MAILCHECK=1800
```

In **csh**, if you want to change how often the shell checks for mail, you need to add the new number of seconds before the pathname in the **MAIL** variable. To tell your **csh** to check for new mail every half hour:

```
set MAIL=(1800 /usr/spool/mail/susannah)
```

Creating command aliases

A command alias is a command you type that stands for a longer, or harder-to-remember, command line. For example, if you are a DOS user, you might create an alias called **dir** instead of trying to remember the **ls** command.

The way you create aliases depends on the shell you are using. In the Bourne shell, you need to set up a shell function, while in the Korn shell and the C shell, you can use the built-in alias command.

To set up an alias in **sh**, add the following lines to your *.profile*:

```
aliasname( ) { command
}
```

Here ***aliasname*** is the name you want to call the alias and ***command*** is the command you want to alias. When you choose a name for your alias, be careful to choose a name that is not already the name of a UNIX command, otherwise, when you type the name of your alias, the UNIX system may think you mean the command of the same name instead.

For example, to create an alias called **dir** in **sh** that shows you a file listing, add the following lines to your *.profile*:

```
dir() { ls
}
```

You can make an alias that uses a filename as an argument, but you need to tell your shell where to insert the filename. You can do this by using the variable **$1**, which the shell reads as "insert the first argument here." For example, if you want to create an alias in **sh** called **print** that runs a file through the **pr** ("pretty print") command before sending it to the lineprinter, you could set up the following function:

```
print() { pr $1 | lp
}
```

To print a file using your new alias, you would type **print *file*** where ***file*** is the name of the file you want printed.

Aliases in the Korn and C shells are introduced by the built-in shell command **alias**. Aliases in the Korn shell have the following format:

```
alias aliasname='command'
```

So, the **dir** alias would be:

```
alias dir='ls'
```

And the **print** alias would be:

```
alias print='pr $1 | lp'
```

Aliases in the C shell have this format:

```
alias aliasname 'command'
```

The **dir** alias in **csh** would be:

```
alias dir 'ls'
```

And the **print** alias in **csh** would be:

```
alias print 'pr -n :* | lp'
```

Summary

Changes made using the following commands affect the current login session only. If you want to change your prompt permanently, for example, you should modify or add the prompt setting command in the appropriate startup file for your shell.

To change your prompt:	In **sh** or **ksh**: `PS1=`***newprompt*** In **csh**: `set prompt=`***newprompt***
To add a directory to your path:	In **sh** or **ksh**: `PATH=$PATH:`***newdir*** In **csh**: `set path=(/bin /usr/bin $HOME/bin` ***newdir*** `.)`
To change the default file permissions:	`umask` ***permsmask***
To change where the shell looks for mail and the new mail message:	In **sh**: `MAILPATH=`***pathname%message*** In **ksh**: `MAILPATH=`***pathname?message*** In **csh**: `set MAILPATH=`***pathname*** (You cannot change the new mail message in **csh**.)
To change how often your shell looks for new mail:	In **sh** or **ksh**: `MAILCHECK=`***seconds*** In **csh**: `set MAIL=(`***seconds pathname***`)`
To create a command alias:	In **sh**: ***aliasname*** `( ) { ` ***command*** `}` In **ksh**: `alias` ***aliasname='command'*** In **csh**: `alias` ***aliasname 'command'***

For more information about	See
File permissions	**chmod**(C)
File creation mask	**umask**(C)
The files your shell reads at startup	Appendix D, "Sample shell startup files" in the *Operating System User's Guide*
The Bourne shell	Chapter 11, "Automating frequent tasks" in the *Operating System User's Guide* **sh**(C)
The Korn shell	Chapter 11, "Automating frequent tasks" in the *Operating System User's Guide* **ksh**(C)
The C shell	Chapter 11, "Automating frequent tasks" in the *Operating System User's Guide* **csh**(C)

Appendix A

Going from DOS to UNIX

This appendix contains a table showing some common MS-DOS commands and their UNIX system equivalents.

For more information about any of the UNIX system commands, consult the *Operating System User's Reference.*

The commands listed in Table A-1, "Equivalent UNIX and DOS commands" (page 100) are for working with UNIX system files. If you have DOS installed on the same machine as your SCO OpenServer system, you can access your DOS files from within the UNIX system. For more information about accessing DOS files from the SCO OpenServer system, see Chapter 6, "Working with DOS" in the *Operating System User's Guide* or **doscmd**(C).

Table A-1 Equivalent UNIX and DOS commands

DOS command	What it does	UNIX system equivalent	Notes
cd	change directories	**cd**(C)	
cls	clear the screen	**clear**(C)	
copy	copy files	**cp**(C), **copy**(C), **tar**(C)	Use **cp** to copy files, **copy** to copy directories, and **tar** to copy files or directories onto floppy disks or tapes.
date	display the system date and time	**date**(C), **cal**(C)	On the UNIX system, **date** displays the date and the time. **cal** displays the date, the time, and a 3-month calendar.
del	delete a file	**rm**(C)	Be careful when using **rm** with wildcard characters, like **rm** *.
dir	list the contents of a directory	**ls**(C)	There are a variety of options to **ls** including **ls -l** to see a long listing, **ls -c** to see a listing in columns, and **ls -f** to see a listing that indicates file types.
diskcomp	make a track-by-track comparison of two floppy disks	**diskcmp**(C)	
diskcopy	copy a source disk to a target disk	**diskcp**(C)	
edlin	line editor	**ed**(C), **ex**(C), **vi**(C)	**vi** is a full-screen text editor with powerful search and replace functions. **ed** and **ex** are predecessors of **vi**.
fc	compare two files	**diff**(C), **diff3**(C), **cmp**(C)	**diff** compares two text files. **diff3** compares three text files. Use **cmp** to compare binary files.
find	find text within a file	**grep**(C)	**grep** (global regular expression parser) finds text within a file. The UNIX system's **find**(C) command finds files.

(Continued on next page)

Table A-1 Equivalent UNIX and DOS commands
(Continued)

DOS command	What it does	UNIX system equivalent	Notes
format	format a disk	**format**(C)	See */etc/default/format* for the default drive to format. The **format** command formats a disk for use with UNIX system files. Use **dosformat** (see **doscmd**(C)) to format a DOS disk.
mkdir	make a directory	**mkdir**(C)	
more	display output one screen at a time	**more**(C)	
print	print files in the background	**lp**(C)	Use **lp** ***filename*** **&** to print in the background. You can run any UNIX system command in the background by adding **&** (ampersand) to the end of the command line.
ren	rename a file	**mv**(C)	
rmdir	remove an empty directory	**rmdir**(C)	Use **rm -r** to remove a directory that is not empty.
sort	sort data	**sort**(C)	
type	display a text file	**cat**(C), **more**(C)	
xcopy	copy directories	**copy**(C), **tar**(C)	Use **tar** if you want to copy directories onto disk or tape.

absolute mode

A method of changing file permissions using 3-digit octal numbers. For example, to add group write permission on a file called *report* using absolute mode, type **chmod 664 report**. Note that you must be *root* or the owner of the file to change permissions on that file. You can also change permissions using symbolic mode.

absolute pathname

A pathname for a file or directory that begins at the *root* directory. Every absolute pathname begins with a slash character (/), which stands for the *root* directory. See also pathname and relative pathname.

application

A computer program that performs a particular task. Word processing, spreadsheet, and database programs are all applications. See also Applications list.

Application folder

A sublist on the main Applications list, which usually includes a list of related application programs. An Application folder can contain applications and other application folders. See also Applications list.

Applications list

The list of available applications and application folders that is displayed on the main SCO Shell screen. See also application and Application folder.

argument

A word you type on the command line that is separated by a space from the command itself. A command can have more than one argument. Arguments tell a command how to you want it to work. For example, **lf -a**; the **-a** option tells the **lf** file listing program that you want it to show *all* files. These types of arguments are also known as options or flags. Arguments can also tell a command what you want it to work on: for example, **lf -a /tmp/spell.out** tells **lf** to list the file */tmp/spell.out* if it exists.

ASCII

The American Standard Code for Information Interchange is a standard way of representing characters on many computer systems. The term "ASCII file" is often used as a synonym for "plain text file," that is, a file without any special formatting, which can be viewed using UNIX system utilities such as **cat**(C), **more**(C), and **vi**(C).

attribute

Attribute bits are set on a file to control which users have permission to read, write, or execute it. See permissions.

Bourne shell

A UNIX system shell, named after its author, Steven R. Bourne. To start a Bourne shell from the command line, type **sh** and press ⟨Enter⟩.

buffer

An area of computer memory used to store information temporarily before it is written out to a more permanent location, like a file.

C shell

An alternative UNIX System V shell supplied with the SCO OpenServer system. This shell, written by William Joy at the University of California at Berkeley, is known for its interactive features, such as the ability to recall and modify previous command lines. The C shell shell programming language has a syntax like that of the C language, hence the name. C shell is the standard shell on older versions of the Berkeley UNIX operating system found at many universities. To start a C shell from the command line, type **csh** and press ⟨Enter⟩.

command alias

An alternative name for a command. When you type the alias, the command is substituted for the alias. Aliases are useful when you remember commands by names other than their UNIX system names; for example, DOS users may think of **dir** rather than **ls** when they want to list a directory. Aliases are also useful for creating commands that perform several UNIX system commands at once. See the *Operating System User's Guide* for more information.

command line

The instructions you type next to the shell prompt. Command lines can contain commands, arguments, and filenames. You can enter more than one command on a command line by joining commands with a pipe (|), or by separating commands using the command separator (;). The shell executes your command line when you press ⟨Enter⟩.

command separator

The semicolon (;) serves as a command separator on the UNIX system. If you want to issue several commands on one line, separate the commands with semicolons before you press ⟨Enter⟩. For example, type **ls; pwd** and press ⟨Enter⟩ to list files and then print the working directory. Commands are executed in sequence as separate processes.

current directory

See current working directory.

current working directory

The directory where you are currently located. Use the **pwd**(C) command (print working directory) to see your current working directory. The current working directory is taken as the starting point for all relative pathnames. This directory is symbolically referred to as "." in directory listings.

device

Peripheral hardware attached to the computer such as a printer, modem, disk or tape drive, terminal, and so on. Devices in the SCO OpenServer system are controlled by device drivers which are linked into the kernel.

directory

Where the UNIX system stores files. Directories in the UNIX system are arranged in an upside-down tree hierarchy, with the root (/) directory at the top. All other directories branch out from the root directory. The UNIX system implements directories as normal files that store the names of the files within them.

environment

The various settings that control the way you work on the UNIX system. These settings are specific to the shell you use and can be modified from the command line or by modifying shell control files. For example, the directories the shell searches to find a command you type are set in the variable **PATH**, which is part of your environment.

environment variable

Special variables that modify your login shell behavior. Typical examples are **PATH**, which defines the directories in which the shell will search for files or commands, and **PROMPT** which determines the on-screen shell prompt message. See also variable.

file

The basic unit of information on a UNIX filesystem. Regular files are usually either text (ASCII) or executable programs. Other types of files exist on the UNIX system such as directories, which store information about the files within them; device files, which are used by the system to access a particular device; and FIFO (First In First Out) pipe files, which are used to transfer data between programs. The attributes of each file are stored in the file's inode. See also directory.

file descriptor

A number associated with an open file; used to refer to the open file in I/O redirection operations.

group

A set of users who are identified with a particular group ID number on the UNIX system. Typically, members of a group are coworkers in a department or on a project. Each file on the UNIX system also has a group associated with it; this group, along with the owner and the permissions controls who can access and modify that file. You can see the group of a file by listing the file with the **l** command. To find out your own group, use the **id**(C) command.

home directory

The place in the filesystem where you can keep your personal files and subdirectories. When you log in, you are automatically placed in your home directory. Typically, this will be */u/**loginname*** or */usr/**loginname***, where ***loginname*** is your login name. The shell's shorthand for the home directory is "~". See tilde expansion.

inode

The internal representation of a file, showing disk layout, owner, type (see file), permissions, access and modification times, size and the number of links. Each inode has a unique decimal identifier.

kernel

The central part of the UNIX operating system, which manages how memory is used, how tasks are scheduled, how devices are accessed, and how file information is stored and updated.

Korn shell

Written by David Korn, it is compatible with the Bourne shell, but provides a much wider range of programming features. The Korn shell also offers improved versions of many of the C shell's interactive features. To start a Korn shell from the command line, type **ksh** and press ⟨Enter⟩. See also Bourne shell and C shell.

link

A filename that points to another file. Links let you access a single file from multiple directories without storing multiple copies of the file. If you make a change to the content of a linked file, the change is reflected in each of the links. All links point to an inode. See also symbolic link.

literal

A literal character or string is one that represents itself, that is, that can be taken literally (as opposed to a pattern, that represents some other characters). For a metacharacter to regain its literal value (for example, for * to mean an asterisk and not "zero or more characters") it must be "quoted". See quoting and wildcard.

log in

The way you gain access to a UNIX system. To log in, you enter your login name and password and the computer verifies these against its user account records before allowing you access.

log out

What you do after you have finished working on a UNIX system. You can log out by pressing ⟨Ctrl⟩**D**, typing **exit**, or typing **logout**, depending on your shell.

login name

The name through which you gain access to the operating system. When you are logging onto the computer, you must enter this login name, followed by a password.

login shell

The shell that is automatically started for you when you log in. You can start to work in other shells, but your login shell will always exist until you log out.

macro

A collection of instructions or keystrokes that may be invoked using a single name or keystroke combination, used to automate regular and complex tasks.

mail alias

A single name used to send mail to several users at once. For example, many users have aliases set up for mailing to the entire company, single departments, or groups of individuals.

manual page

An entry in a UNIX reference manual. These entries can be accessed online using the **man**(C) command. A letter in parentheses following a command or filename refers to the reference manual section where the command or file is documented. For example, the **man**(C) command is documented in section C, Commands, of the *Operating System User's Reference.* They are also called "man pages."

mask

A series of bit settings that "cover up" existing settings, only allowing some settings to show through, while masking out others.

metacharacter

A special character that is replaced by matching character strings when interpreted by the shell. Metacharacters, which define the form of a string, and literal characters, which match only themselves, make up regular expressions.

multitasking

A system that can do several jobs at once.

multiuser

A system that can be used by more than one person at the same time.

named buffer

A buffer used to copy text between files in the **vi**(C) editor. **vi** clears unnamed buffers when it switches files, but the contents of named buffers are preserved.

online

Accessible from your terminal screen.

operating system

A group of programs that provide basic functionality on a computer. These programs operate your computer hardware in response to commands like **copy** and **print**, and form a set of functional building blocks upon which other programs depend. An operating system also manages computer resources such as peripheral devices like disk drives or printers attached to the computer and resolves resource conflicts, as when two programs want to use a disk drive at the same time.

owner

1. The user who created a file or directory. Only the owner and *root* can change the permissions assigned to the file or directory.

2. One of the attributes of a file that, along with its group and permissions, determine who can access and modify that file. You can see the owner of a file by listing it with the **l** command. Use the **chown**(C) command to change the owner of a file.

password

The string of characters you are prompted for after you type your login name when you are logging in. Your password is the key that lets you into the UNIX system; you should choose it wisely, keep it secret, and change it regularly. Use the **passwd**(C) command to change your password.

path

The directory list through which your shell searches to find the commands you type. Your path is stored in the shell variable **PATH**.

pathname

The name of a directory or a file, for example, */usr/spool/mail*. Each component of a pathname, as separated by slashes, is a directory, except for the last component of a pathname, which can be either a directory or a file. A single word by itself, such as *Tutorial*, can be a pathname; this is a relative pathname for the file or directory *Tutorial* from the current working directory. A single slash, (/), is the pathname for the root directory. See also absolute pathname and relative pathname.

permissions

The settings (also called properties or attributes) associated with each file or directory that determine who can access or modify the file and directory. Use the **l** command to list a file's permissions; use the **chmod**(C) (change mode) command to change a file's permissions.

pipe

A way of joining commands on the command line so that the output of one command provides the input for the next. To use a pipe on the command line, join commands with the vertical bar symbol, (|). For example, to sort a file, eliminate duplicate lines, and print it, you could type **sort *file* | uniq | lp**.

print job

A request you have made to the printer to print a file. Each print job has an ID number that you can see using the **lpstat**(C) command. You can cancel a print job by typing **cancel** and its job ID number, then pressing ⟨Enter⟩.

process ID

A number that uniquely identifies a running program on the UNIX system. This is also known as the PID.

prompt

One or more characters or symbols that identify a line on which commands can be entered, as in a UNIX or DOS window. "Prompt" also refers to the text displayed when the computer displays a request for input or an instruction. The default prompt can be replaced by setting the PS1 environment variable.

quoting

A mechanism that is used to control the substitution of special characters. Special characters enclosed in single quotes are not replaced by their meaning, but remain embedded in the text when the quotes are stripped off. Double quotes are used to prevent the expansion of all special characters except " $ ", " \ " and " ` ".

regular expression

A notation for matching any sequence of characters. The notation is used to describe the *form* of a sequence of characters, rather than the characters themselves. Regular expressions consist of literal characters, which match only themselves, and metacharacters.

relative pathname

A pathname that does not start with a slash (/); for example; *Tutorial, Reports/September*, or *../tmp*. A relative pathname is searched for, starting from the current working directory and may use the notation " .. " to indicate "one directory up from the current working directory." See also absolute pathname and pathname.

root

The top directory of a UNIX filesystem, represented as a slash (/). Also, the login name of the superuser, a user who has the widest form of computer privileges.

shell

A program that controls how the user interacts with the operating system. Using such programs, you can write a shell script to automate work you do regularly. The shells available with the SCO OpenServer system include the Korn shell, the Bourne shell, and the C shell.

shell escape

A command you type from within an interactive program to escape to the shell. In **vi**, you can type **:!*command*** to escape to the shell and execute ***command***. When ***command*** has finished executing, you are returned to the editor. You can start a new shell this way with **:!sh**, for example. To exit this subshell and return to the editor, press ⟨Ctrl⟩**D** or type **exit**.

shell programming language

A programming language that is built into the shell. The Korn shell, the Bourne shell, and the C shell all have slightly different programming languages but all three shells offer basics such as variable creation, loops, and conditional tests.

shell script

An executable text file written in a shell programming language. Scripts are made up of shell programming commands mixed with regular UNIX system commands. To run a shell script, you can change its permissions to make it an executable file, or you can use it as the argument to a shell command line (for example, **sh *script***). The shell running the script will read it one line at a time and perform the requested commands.

shell variable

A variable associated with a shell script.

standard error

The usual place where a program writes its error messages. By default, this is the screen. Standard error can be redirected; to a file, for example. Also known as stderr.

standard input

The usual place from which a program takes its input. By default, this is the keyboard. Standard input can be redirected; for example, you can use the less-than symbol (<) to instruct a program to take input from a file. Also known as stdin, the standard input is identified by the file descriptor 0.

standard output

The usual place where a program writes its output. By default, this is the screen. Standard output can be redirected; for example, you can use a pipe symbol (|) to instruct a program to write its output into a pipe, which will then be read as input by the next program in the pipeline. Also known as stdout, the standard output is identified by the file descriptor 1.

superuser

A user who has powerful special privileges needed to help administer and maintain the system. The superuser logs in as *root*. Someone with the superuser or *root* password can access and modify any file on the system.

symbolic link

A new name that refers to a directory or file that already exists. Use this name to change to another directory without typing its full pathname. Unlike normal links, symbolic links can cross filesystems and link to directories. See also link.

symbolic mode

A method of changing file permissions using keyletters to specify which set of permissions to change and how to change them. For example, to add group write permission on a file called *report* using symbolic mode, you could type **chmod g+w report**. Note that you must be the owner of a file or the superuser to change permissions on that file. You can also change permissions using absolute mode.

system administrator

The person who looks after the day-to-day running of the computer and performs tasks such as setting up user accounts and making system backups.

terminal

Video display unit with a keyboard, a monitor, and sometimes a mouse. They do not have any independent processing power themselves and they must be connected to a computer before they can do any useful work.

terminal type

A name for the kind of terminal from which you are working. Typically, the terminal type is an abbreviation of the make and model of the terminal, such as *wy60*, which is the terminal type for a Wyse60. Your terminal type is stored in the variable **TERM**.

tilde expansion

The ability of the shell to translate instances of the tilde character (~) into the pathname of the user's home directory.

umask

A permissions mask that controls the permissions assigned to new files you create. You can set your umask from the command line or in one of your shell startup files.

user account

The records a UNIX system keeps for each user on the system.

variable

An object known to your shell that stores a particular value. The value of a variable can be changed either from inside a program or from the command line. Each shell variable controls a particular aspect of your working environment on the UNIX system. For example, the variable **PS1** stores your primary prompt string.

wildcard

A character (such as "?" or "*") that is substituted with another character or a group of characters in text searches and similar operations. See also meta-character.

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

1 May 1995

AU20002P001