

Digital UNIX

Command and Shell User's Guide

Order Number: AA-PS2HD-TE

March 1996

Product Version: Digital UNIX Version 4.0 or higher

This book explains to the general Digital UNIX user how to use commands and shells and how to communicate with other network users.

Digital Equipment Corporation
Maynard, Massachusetts

Digital Equipment Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Digital or an authorized sublicensor.

© Digital Equipment Corporation 1996
All rights reserved.

The following are trademarks of Digital Equipment Corporation:

ALL-IN-1, Alpha AXP, AlphaGeneration, AlphaServer, AlphaStation, AXP, Bookreader, CDA, DDIS, DEC, DEC Ada, DEC Fortran, DEC FUSE, DECnet, DECstation, DECsystem, DECterm, DECUS, DECwindows, DTIF, MASSBUS, MicroVAX, OpenVMS, POLYCENTER, Q-bus, StorageWorks, TruCluster, TURBOchannel, ULTRIX, ULTRIX Mail Connection, ULTRIX Worksystem Software, UNIBUS, VAX, VAXstation, VMS, XUI, and the DIGITAL logo.

NFS is a registered trademark of Sun Microsystems, Inc. Open Software Foundation, OSF, OSF/1, OSF/Motif, and Motif are trademarks of the Open Software Foundation, Inc. UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Ltd.

All other trademarks and registered trademarks are the property of their respective holders.

Contents

About This Book

Audience	xix
Scope	xix
Organization	xx
Related Documents	xxi
Reader's Comments	xxii
Conventions	xxiii

1 Getting Started

1.1 Logging In	1-1
1.2 Logging Out	1-4
1.3 Using Commands	1-4
1.4 Stopping Command Execution	1-6
1.5 Setting Your Password	1-6
1.5.1 Password Guidelines	1-7
1.5.2 Password Procedures	1-8
1.6 Getting Help	1-9
1.6.1 Displaying and Printing Online Reference Pages (man)	1-9
1.6.2 Locating Commands Using Descriptive Keywords	1-10

2 Overview of Files and Directories

2.1	Overview of Text Editors	2-1
2.2	Creating Sample Files with the vi Text Editor	2-2
2.3	Understanding Files, Directories, and Pathnames	2-5
2.3.1	Files and Filenames	2-6
2.3.2	Directories and Subdirectories	2-7
2.3.3	Displaying the Name of Your Current (Working) Directory (pwd)	2-7
2.3.4	The Tree-Structure File System and Pathnames	2-8
2.4	Specifying Files with Pattern Matching	2-13

3 Managing Files

3.1	Listing Files (ls)	3-2
3.1.1	Listing Contents of the Current Directory	3-2
3.1.2	Listing Contents of Other Directories	3-3
3.1.3	Flags Used with the ls Command	3-3
3.2	Displaying Files	3-5
3.2.1	Displaying Files Without Formatting (pg, more, cat)	3-6
3.2.2	Displaying Files with Formatting (pr)	3-7
3.3	Printing Files (lpr, lpq, lprm)	3-10
3.4	Linking Files (ln)	3-13
3.4.1	Hard Links and Soft Links	3-14
3.4.2	Links and File Systems	3-14
3.4.3	Using Links	3-15
3.4.4	How Links Work – Understanding Filenames and i-numbers ..	3-16
3.4.5	Removing Links	3-17
3.5	Copying Files (cp)	3-19
3.5.1	Copying Files in the Current Directory	3-20
3.5.2	Copying Files into Other Directories	3-21

3.6	Renaming or Moving Files (mv)	3-22
3.6.1	Renaming Files	3-22
3.6.2	Moving Files into a Different Directory	3-23
3.7	Comparing Files (diff)	3-24
3.8	Sorting File Contents (sort)	3-26
3.9	Removing Files (rm)	3-27
3.9.1	Removing a Single File	3-27
3.9.2	Removing Multiple Files – Matching Patterns	3-28
3.10	Determining File Type (file)	3-30

4 Managing Directories

4.1	Creating a Directory (mkdir)	4-1
4.2	Changing Directories (cd)	4-4
4.2.1	Changing Your Current Directory	4-4
4.2.2	Using Relative Pathname Notation	4-5
4.2.3	Accessing Directories Through Symbolic Links	4-7
4.3	Displaying Directories (ls -F)	4-8
4.4	Copying Directories (cp)	4-8
4.5	Renaming Directories (mv)	4-10
4.6	Removing Directories (rmdir)	4-10
4.6.1	Removing Empty Directories	4-11
4.6.2	Removing Multiple Directories	4-12
4.6.3	Removing Your Current Directory	4-12
4.6.4	Removing Files and Directories Simultaneously (rm -r)	4-13

5 Controlling Access to Your Files and Directories

5.1	Understanding Password and Group Security Files	5-1
5.1.1	The /etc/passwd File	5-2

5.1.2	The /etc/group File	5-3
5.2	Protecting Files and Directories	5-3
5.3	Displaying File and Directory Permissions (ls)	5-5
5.4	Setting File and Directory Permissions (chmod)	5-8
5.4.1	Specifying Permissions with Letters and Operation Symbols ..	5-8
5.4.1.1	Changing File Permissions	5-9
5.4.1.2	Changing Directory Permissions	5-10
5.4.1.3	Using Pattern-Matching Characters	5-10
5.4.1.4	Setting Absolute Permissions	5-11
5.4.2	Specifying Permissions with Octal Numbers	5-11
5.5	Setting Default Permissions with the User Mask (umask)	5-13
5.5.1	Setting the umask	5-15
5.6	Changing Your Identity to Access Files	5-17
5.7	Superuser Concepts	5-18
5.8	Changing Owners and Groups (chown and chgrp)	5-19
5.9	Additional Security Considerations	5-20
6	Using Processes	
6.1	Understanding Programs and Processes	6-1
6.2	Understanding Standard Input, Output, and Error	6-1
6.2.1	Redirecting Input and Output	6-2
6.2.1.1	Reading Input from a File	6-2
6.2.1.2	Redirecting Output	6-3
6.2.2	Redirecting Standard Error to a File	6-4
6.2.2.1	Bourne and Korn Shell Error Redirection	6-4
6.2.2.2	C Shell Error Redirection	6-5
6.2.3	Redirecting Both Standard Error and Standard Output	6-5
6.3	Running Several Processes Simultaneously	6-6

6.3.1	Running Foreground Processes	6-6
6.3.2	Running Background Processes	6-7
6.4	Monitoring and Terminating Processes	6-8
6.4.1	Checking Process Status	6-9
6.4.1.1	The ps Command	6-9
6.4.1.2	The jobs Command	6-11
6.4.2	Canceling a Foreground Process (Ctrl/C)	6-11
6.4.3	Canceling a Background Process (kill)	6-12
6.4.4	Suspending and Resuming a Foreground Process (C Shell Only)	6-13
6.5	Displaying Information About Users and Their Processes	6-14

7 Shell Overview

7.1	Purpose of Shells	7-1
7.2	Summary of Bourne, C, and Korn Shell Features	7-2
7.2.1	More Information on C and Korn Shell Features	7-3
7.2.2	The Restricted Bourne Shell	7-4
7.3	Changing Your Shell	7-4
7.3.1	Determining What Shell You Are Running	7-5
7.3.2	Temporarily Changing Your Shell	7-5
7.3.3	Permanently Changing Your Shell	7-6
7.4	Command Entry Aids	7-6
7.4.1	Using Multiple Commands and Command Lists	7-7
7.4.1.1	Running Commands in Sequence with a Semicolon (;)	7-7
7.4.1.2	Running Commands Conditionally	7-8
7.4.2	Using Pipes and Filters	7-8
7.4.3	Grouping Commands	7-10
7.4.3.1	Using Parentheses ()	7-11
7.4.3.2	Using Braces { }	7-11
7.4.4	Quoting	7-11

7.4.4.1	Using the Backslash (\)	7-12
7.4.4.2	Using Single Quotes (' ')	7-12
7.4.4.3	Using Double Quotes (" ")	7-13
7.5	The Shell Environment	7-13
7.5.1	The login Program	7-13
7.5.2	Environment Variables	7-14
7.5.3	Shell Variables	7-16
7.6	Login Scripts and Your Environment	7-16
7.7	Using Variables	7-18
7.7.1	Setting Variables	7-19
7.7.1.1	Bourne and Korn Shell Variables	7-19
7.7.1.2	C Shell Variables	7-20
7.7.1.3	Setting Variables in All Shells	7-21
7.7.2	Referencing Variables (Parameter Substitution)	7-21
7.7.3	Displaying the Values of Variables	7-21
7.7.4	Clearing the Values of Variables	7-22
7.8	How the Shell Finds Commands	7-23
7.9	Using Logout Scripts	7-24
7.9.1	Logout Scripts and the Shell	7-25
7.9.2	A Sample .logout File	7-25
7.10	Using Shell Procedures (Scripts)	7-26
7.10.1	Writing and Running Shell Procedures	7-26
7.10.2	Specifying a Run Shell	7-28
8	Shell Features	
8.1	Comparison of C, Bourne, and Korn Shell Features	8-1
8.2	C Shell Features	8-3
8.2.1	Sample .cshrc and .login Scripts	8-3
8.2.2	Metacharacters	8-6
8.2.3	Command History	8-8

8.2.4	Filename Completion	8-9
8.2.5	Aliases	8-10
8.2.6	Built-In Variables	8-11
8.2.7	Built-In Commands	8-12
8.3	Bourne Shell Features	8-14
8.3.1	Sample .profile Login Script	8-14
8.3.2	Metacharacters	8-16
8.3.3	Built-In Variables	8-17
8.3.4	Built-In Commands	8-18
8.4	Korn Shell Features	8-19
8.4.1	Sample .profile and .kshrc Login Scripts	8-20
8.4.2	Metacharacters	8-24
8.4.3	Command History	8-25
8.4.4	Command Line Editing Using the fc Command	8-27
8.4.4.1	Examples of Command Line Editing	8-28
8.4.5	Filename Completion	8-29
8.4.6	Aliases	8-30
8.4.7	Built-In Variables	8-32
8.4.8	Built-In Commands	8-33

9 Using the System V Habitat

9.1	Setting Up Your Environment	9-2
9.2	How the System V Habitat Access Works	9-3
9.3	Compatibility for Shell Scripts	9-4
9.4	System V Habitat Command Summary	9-4

10 Obtaining Information About Network Users and Hosts

10.1	Identifying Users on the Local Host	10-1
10.2	Obtaining Information About Network Users	10-2

10.2.1	Obtaining Information About a Specific User	10-3
10.2.2	Obtaining Information About Users on a Remote Host	10-4
10.2.3	Obtaining Information About an Individual User on a Remote Host	10-4
10.2.4	Customizing Output from the finger Command	10-4
10.3	Obtaining Information about Remote Hosts and Users	10-5
10.4	Obtaining Information About Users on Remote Hosts	10-8
10.5	Determining Whether a Remote Host Is Online	10-9

11 Sending and Receiving Messages

11.1	Addressing Mail Messages	11-1
11.2	Sending a Mail Message Using mailx	11-2
11.2.1	Editing a Message	11-4
11.2.2	Aborting a Message	11-4
11.2.2.1	Aborting a Message with Ctrl/c	11-5
11.2.2.2	Aborting a Message with an Escape Command	11-5
11.2.3	Including a File within a Message	11-6
11.3	Receiving a Mail Message	11-7
11.3.1	Deleting a Message	11-9
11.3.2	Replying to a Message	11-10
11.3.3	Saving a Message	11-11
11.3.3.1	Saving a Message in a File	11-11
11.3.3.2	Saving a Message in a Folder	11-12
11.3.4	Forwarding a Message	11-14
11.4	Getting Help from mailx	11-14
11.5	Exiting Mail	11-15
11.6	Customizing Mail Sessions	11-15
11.6.1	Creating Mail Aliases	11-16
11.6.2	Setting Mail Variables	11-16
11.7	The Message Handling (MH) Program	11-17

11.8	Sending and Receiving Messages With write	11-21
11.9	Sending and Receiving Messages with talk	11-23

12 Copying Files to Another Host

12.1	Copying Files Between a Local and a Remote Host	12-1
12.1.1	Using rcp to Copy Files Between Local and Remote Hosts .	12-2
12.1.2	Using ftp to Copy Files Between Local and Remote Hosts .	12-3
12.1.3	Using mailx to Copy ASCII Files Between Local and Remote Hosts	12-9
12.1.4	Using write to Copy Files Between Local and Remote Hosts .	12-9
12.2	Copying Directories of Files Between a Local and a Remote Host .	12-9
12.3	Copying Files Between Two Remote Hosts	12-10

13 Working on a Remote Host

13.1	Using rlogin to Log in to a Remote Host	13-1
13.2	Using rsh to Run Commands on a Remote Host	13-2
13.3	Using telnet to Log Into a Remote Host	13-3

14 The UUCP Networking Commands

14.1	UUCP Pathname Conventions	14-1
14.2	Finding Hosts that Support UUCP	14-2
14.3	Connecting to a Remote Host	14-2
14.3.1	Using cu to Connect to a Remote Host	14-3
14.3.1.1	Using cu to Connect by Name to a Remote Host	14-3
14.3.1.2	Using cu to Specify a Directly-Connected Remote Host	14-4
14.3.1.3	Using cu to Connect by Telephone to a Remote Host .	14-4
14.3.1.4	Local cu Commands	14-6
14.3.1.5	Using cu to Connect a Local Computer to Several Remote Computers	14-8

14.3.2	Using tip to Connect to a Remote Host	14-9
14.3.2.1	Using tip to Connect by Name to a Remote Host	14-9
14.3.2.2	Using tip to Connect by Telephone to a Remote Host .	14-10
14.3.2.3	Local tip Commands	14-12
14.3.2.4	Using tip to Connect a Local Computer to Several Remote Computers	14-13
14.3.3	Using ct to Connect to a Remote Terminal with a Modem ..	14-15
14.4	Using uux to Run Commands on Remote Hosts	14-18
14.4.1	Using uux from the Bourne or Korn Shells	14-19
14.4.2	Using uux from the C Shell	14-20
14.4.3	Other uux Features and Suggestions	14-20
14.5	Using UUCP To Send and Receive Files	14-21
14.5.1	Using UUCP to Copy Files in the Bourne and Korn Shells .	14-22
14.5.2	Using UUCP to Copy Files in the C Shell	14-23
14.6	Using uuto with uupick to Copy Files	14-24
14.7	Using uuto to Send a File Locally	14-26
14.8	Displaying Job Status of UUCP Utilities	14-27
14.8.1	The uustat Command	14-27
14.8.1.1	Displaying the Holding Queue Output with a uustat Option	14-28
14.8.1.2	Displaying the Current Queue Output with uustat Options	14-29
14.8.2	Using the uulog Command to Display UUCP Log Files	14-31
14.8.3	Monitoring UUCP Status	14-32

A A Beginner's Guide to Using vi

A.1	Getting Started	A-2
A.1.1	Creating a File	A-2
A.1.2	Opening an Existing File	A-4
A.1.3	Saving a File and Quitting vi	A-4
A.1.4	Moving Within a File	A-6
A.1.4.1	Moving the Cursor Up, Down, Left, and Right	A-6

A.1.4.2	Moving the Cursor by Word, Line, Sentence, and Paragraph	A-7
A.1.4.3	Moving and Scrolling the Cursor Forward and Backward Through a File	A-8
A.1.4.4	Movement Command Summary	A-9
A.1.5	Entering New Text	A-10
A.1.6	Editing Text	A-13
A.1.6.1	Deleting Words	A-13
A.1.6.2	Deleting Lines	A-14
A.1.6.3	Changing Text	A-14
A.1.6.4	Text Editing Command Summary	A-14
A.1.7	Undoing a Command	A-15
A.1.8	Finishing Your Edit Session	A-16
A.2	Using Advanced Techniques	A-16
A.2.1	Searching for Strings	A-16
A.2.2	Deleting and Moving Text	A-17
A.2.3	Yanking and Moving Text	A-18
A.2.4	Other vi Features	A-18
A.3	Using the Underlying ex Commands	A-18
A.3.1	Making Substitutions	A-20
A.3.2	Writing a Whole File or Parts of a File	A-21
A.3.3	Deleting a Block of Text	A-22
A.3.4	Customizing Your Environment	A-22
A.3.5	Saving Your Customizations	A-24

B Creating and Editing Files with ed

B.1	Understanding Text Files and the Edit Buffer	B-1
B.2	Creating and Saving Text Files	B-2
B.2.1	Starting the ed Program	B-2
B.2.2	Entering Text – The a (Append) Subcommand	B-2
B.2.3	Displaying Text – The p (print) Subcommand	B-3
B.2.4	Saving Text – The w (write) Subcommand	B-4
B.2.4.1	Saving Text under the Same Filename	B-4
B.2.4.2	Saving Text under a Different Filename	B-5

B.2.4.3	Saving Part of a File	B-5
B.2.5	Leaving the ed Program – The q (quit) Subcommand	B-6
B.3	Loading Files into the Edit Buffer	B-6
B.3.1	Using the ed (edit) Command	B-7
B.3.2	Using the e (Edit) Subcommand	B-7
B.3.3	Using the r (read) Subcommand	B-8
B.4	Displaying and Changing the Current Line	B-9
B.4.1	Finding Your Position in the Buffer	B-10
B.4.2	Changing Your Position in the Buffer	B-10
B.5	Locating Text	B-12
B.5.1	Searching Forward Through the Buffer	B-12
B.5.2	Searching Backward Through the Buffer	B-13
B.5.3	Changing the Direction of a Search	B-13
B.6	Making Substitutions – The s (substitute) Subcommand	B-13
B.6.1	Substituting on the Current Line	B-14
B.6.2	Substituting on a Specific Line	B-15
B.6.3	Substituting on Multiple Lines	B-15
B.6.4	Changing Every Occurrence of a String	B-15
B.6.5	Removing Characters	B-16
B.6.6	Substituting at Line Beginnings and Ends	B-16
B.6.7	Using a Context Search	B-17
B.7	Deleting Lines – The d (delete) Subcommand	B-18
B.7.1	Deleting the Current Line	B-18
B.7.2	Deleting a Specific Line	B-19
B.7.3	Deleting Multiple Lines	B-19
B.8	Moving Text – The m (move) Subcommand	B-19
B.9	Changing Lines of Text – The c (change) Subcommand	B-20
B.9.1	Changing a Single Line of Text	B-21
B.9.2	Changing Multiple Lines of Text	B-21
B.10	Inserting Text – The i (insert) Subcommand	B-22

B.10.1	Using Line Numbers	B-22
B.10.2	Using a Context Search	B-23
B.11	Copying Lines – The t (transfer) Subcommand	B-23
B.12	Using System Commands from ed	B-24
B.13	Ending the ed Program	B-25

C Using Internationalization Features

C.1	Understanding Locale	C-1
C.2	How Locale Affects Processing and Display of Data	C-2
C.2.1	Collation	C-3
C.2.2	Date and Time Formats	C-4
C.2.3	Numeric and Monetary Formats	C-5
C.2.4	Messages	C-5
C.2.5	Yes/No Prompts	C-5
C.3	Determining Whether a Locale Has Been Set	C-5
C.4	Setting a Locale	C-6
C.4.1	Locale Categories	C-8
C.4.2	Limitations of Locale Settings	C-10
C.4.2.1	Locale Settings Are Not Validated	C-10
C.4.2.2	File Data Is Not Bound to a Locale	C-10
C.4.2.3	Setting LC_ALL Overrides All Other Locale Variables	C-10

D Customizing Your mailx Session

E Using Escape Commands in Your mailx Session

F Using the mailx Commands

Index

Examples

11-1: Output from mailx Help Command	11-14
12-1: Using ftp to Copy a File	12-4

Figures

1-1: Shell Interaction with the User and the Operating System	1-5
2-1: A Typical Digital UNIX File System	2-9
2-2: Relative and Full Pathnames	2-12
3-1: Removing Links and Files	3-18
4-1: Relationship Between Directories and Subdirectories	4-3
4-2: Copying a Directory Tree	4-9
5-1: File and Directory Permission Fields	5-7
7-1: Flow Through a Pipeline	7-9
9-1: System V Habitat	9-2

Tables

3-1: The ls Command Options	3-4
3-2: The ls -l Command Information	3-5
3-3: The pr Command Flags	3-8
3-4: The lpr Command Flags	3-11
5-1: Differences Between File and Directory Permissions	5-5
5-2: Permission Combinations	5-12
5-3: How Octal Numbers Relate to Permission Fields	5-13

5-4: The umask Permission Combinations	5-14
6-1: Shell Notation for Reading Input and Redirecting Output	6-2
7-1: Shell Names and Default Prompts	7-5
7-2: Multiple Command Operators	7-7
7-3: Command Grouping Symbols	7-10
7-4: Shell Quoting Conventions	7-12
7-5: Selected Shell Environment Variables	7-14
7-6: System and Local Login Scripts	7-17
7-7: Description of Example Shell Script	7-27
8-1: C, Bourne, and Korn Shell Features	8-1
8-2: Example .cshrc Script	8-4
8-3: Example .login Script	8-5
8-4: C Shell Metacharacters	8-6
8-5: Reexecuting History Buffer Commands	8-9
8-6: Built-In C Shell Variables	8-12
8-7: Built-In C Shell Commands	8-13
8-8: Example Bourne Shell .profile Script	8-15
8-9: Bourne Shell Metacharacters	8-16
8-10: Built-In Bourne Shell Variables	8-18
8-11: Built-In Bourne Shell Commands	8-19
8-12: Example Korn Shell .profile Script	8-21
8-13: Example .kshrc Script	8-23
8-14: Korn Shell Metacharacters	8-24
8-15: Reexecuting History Buffer Commands	8-26
8-16: Built-In Korn Shell Variables	8-32
8-17: Built-In Korn Shell Commands	8-33
9-1: User Commands Summary	9-5
10-1: Options to the finger Command	10-4

10-2: Options to the ruptime Command	10-7
11-1: Commands for the MH Message-Handling Program	11-19
12-1: ftp Subcommands for Connecting to a Host and Copying Files	12-5
12-2: ftp Subcommands for Directory and File Modification	12-7
12-3: ftp Subcommands for Help and Status Information	12-8
13-1: telnet Subcommands	13-4
14-1: Options to the cu Command	14-5
14-2: Local cu Commands	14-8
14-3: Options to the tip Command	14-12
14-4: Local tip Commands	14-14
14-5: Options to the ct Command	14-17
14-6: Options to the uux Command	14-21
14-7: Options to the UUCP Command	14-24
14-8: Options to the uupick Command	14-25
14-9: Options to the uuto Command	14-26
14-10: Options to the uustat Command	14-30
14-11: Options to the uulog Command	14-31
A-1: Write and Quit Command Summary	A-6
A-2: Cursor Movement Command Summary	A-9
A-3: Text Insertion Command Summary	A-12
A-4: Text Editing Command Summary	A-15
A-5: Selected vi Environment Variables	A-22
C-1: Locale Names	C-7
C-2: Environment Variables That Influence Locale Functions	C-8
D-1: Variables for Customizing Your mailx Session	D-1
E-1: Escape Commands in mail	E-1
F-1: Commands for the mailx Program	F-1

About This Book

Command and Shell User's Guide introduces users to the basic use of commands and shells in the Digital UNIX® operating system. This book also documents how to communicate with other network users.

This preface covers the following topics:

- Audience
- Scope
- Organization
- Related Documents
- Conventions

Audience

This book is written for those who do not have extensive knowledge of UNIX compatible operating systems. This book explains important concepts, provides tutorials, and is organized according to task.

Scope

This book introduces you to the use of commands and shells. After reading this book, you should be able to:

- Gain access to your system and issue commands
- Understand file and directory concepts
- Manage files and directories
- Control access to your files and directories
- Manage processes
- Understand and manage your shell environment
- Use the `vi` and `ed` text editors
- Use network applications to communicate with network users and access remote systems and processes

Organization

This book is organized into 13 chapters and 6 appendices:

- Chapter 1 Shows how to log in and out of your system, enter commands, set your password, and obtain online help.
- Chapter 2 Provides an overview of the file system, consisting of the files and directories that are used to store text, programs, and other data. This chapter also introduces you to the `vi` text editor, a program that allows you to create and modify files.
- Chapter 3 Shows how to manage files. You will learn how to list, display, copy, move, link, and remove them.
- Chapter 4 Explains how to manage directories. You will learn how to create, change, display, copy, rename, and remove them.
- Chapter 5 Shows how to control access to your files and directories by setting appropriate permissions. It also describes standard password and group security issues as well as provides an overview of additional security considerations.
- Chapter 6 Describes how the operating system creates and keeps track of processes. This chapter explains how to redirect process input, output, and error information, run processes simultaneously, display process information, and cancel processes.
- Chapter 7 Introduces features common to the three shells available with the operating system: the Bourne, C, and Korn shells. You learn how to change your shell, use command entry aids, understand some features of your shell environment (login scripts, environment and shell variables), set and clear variables, write logout scripts, and write and run basic shell procedures.
- Chapter 8 Provides detailed reference information about the C, Bourne, and Korn shells, comparing their features. It details the commands and environment variables of each program and shows how to set up your login script.
- Chapter 9 Shows how to access the System V habitat, a subset of commands, subroutines, and system calls that conforms to the System V Interface Definition (SVID).
- Chapter 10 Provides information on how to get information about other users and remote hosts on the network.
- Chapter 11 Provides information about how to send a message to another user.

- Chapter 12 Provides information about how to copy files to or between remote hosts.
- Chapter 13 Provides information about how to log in to or execute commands at a remote host.
- Chapter 14 Provides information about the UNIX-to-UNIX Copy Program (UUCP) for performing communication tasks concurrently on both a local and remote host.

The appendixes in this book provide the following information:

- Appendix A Teaches you how to use the basic features of the `vi` text editor.
- Appendix B Teaches you how to use the `ed` text editor. Detailed information about `ed` is provided because all systems have this editor, and `ed` can be used in critical system management situations when no other editor can be used.
- Appendix C Describes the internationalization features that allow users to process data and interact with the system in a manner appropriate to their native language, customs, and geographic region.
- Appendix D Provides a list of variables that can be used in the `.mailrc` file to customize a `mailx` session.
- Appendix E Provides a list of escape commands that can be used to perform certain tasks from within a `mailx` session.
- Appendix F Provides a list of commands that can be used to send, read, delete, or save messages using `mailx`.

Related Documents

The following Digital UNIX user documents are available in Bookreader format on your CD-ROM and optionally in hardcopy:

- *Quick Reference Card*
- *Documentation Overview, Glossary, and Master Index*
- *System Administration*
- *Reference Pages Section 1*
- *Reference Pages Section 8 and 1m*

The following OSF/1 documents are currently available from Prentice-Hall:

- *OSF/1 Command Reference*
- *OSF/1 System and Network Administrator's Reference*

The printed version of the Digital UNIX documentation set is color coded to help specific audiences quickly find the books that meet their needs. (You can order the printed documentation from Digital.) This color coding is reinforced with the use of an icon on the spines of books. The following list describes this convention:

Audience	Icon	Color Code
General users	G	Blue
System and network administrators	S	Red
Programmers	P	Purple
Device driver writers	D	Orange
Reference page users	R	Green

Some books in the documentation set help meet the needs of several audiences. For example, the information in some system books is also used by programmers. Keep this in mind when searching for information on specific topics.

The *Documentation Overview*, *Glossary*, and *Master Index* provides information on all of the books in the Digital UNIX documentation set.

Reader's Comments

Digital welcomes any comments and suggestions you have on this and other Digital UNIX manuals.

You can send your comments in the following ways:

- Fax: 603-881-0120 Attn: UEG Publications, ZK03-3/Y32
- Internet electronic mail: readers_comment@zk3.dec.com

A Reader's Comment form is located on your system in the following location:

```
/usr/doc/readers_comment.txt
```

- Mail:

Digital Equipment Corporation
UEG Publications Manager
ZK03-3/Y32
110 Spit Brook Road
Nashua, NH 03062-9987

A Reader's Comment form is located in the back of each printed manual. The form is postage paid if you mail it in the United States.

Please include the following information along with your comments:

- The full title of the book and the order number. (The order number is printed on the title page of this book and on its back cover.)
- The section numbers and page numbers of the information on which you are commenting.
- The version of Digital UNIX that you are using.
- If known, the type of processor that is running the Digital UNIX software.

The Digital UNIX Publications group cannot respond to system problems or technical support inquiries. Please address technical questions to your local system vendor or to the appropriate Digital technical support office. Information provided with the software media explains how to send problem reports to Digital.

Conventions

The following conventions are used in this book:

%	A percent sign represents the C shell system prompt. A dollar sign
\$	represents the system prompt for the Bourne and Korn shells.
#	A number sign represents the superuser prompt.
% cat	Boldface type in interactive examples indicates typed user input.
<i>file</i>	Italic (slanted) type indicates variable values, placeholders, and function argument names.
[]	In syntax definitions, brackets indicate items that are optional and
{ }	braces indicate items that are required. Vertical bars separating items
	inside brackets or braces indicate that you choose one item from
	among those listed.
. . .	In syntax definitions, a horizontal ellipsis indicates that the preceding
	item can be repeated one or more times.
cat(1)	A cross-reference to a reference page includes the appropriate section
	number in parentheses. For example, <code>cat(1)</code> indicates that you can
	find information on the <code>cat</code> command in Section 1 of the reference
	pages.
Ctrl/x	This symbol indicates that you hold down the first named key while
	pressing the key or mouse button that follows the slash. In examples,
	this key combination is enclosed in a box (for example, <code>Ctrl/C</code>).

Getting Started 1

This chapter introduces the basic tasks for using the operating system. Before reading this chapter, familiarize yourself with your system's hardware components.

If you are familiar with the UNIX operating system or other operating systems, you may want to skim this chapter.

After completing this chapter, you will be able to:

- Log in to and log out of the operating system
- Set and change your password
- Execute commands
- Stop command execution
- View and display reference (man) pages

To use the operating system to its full capabilities, you must learn how to create and modify files with a text editing program. See Chapter 2 for an overview of text editors, and Appendix A and Appendix B for information on the `vi` and `ed` text editors, respectively. Once you learn how to use a text editor, you should have the basic skills necessary to start using the operating system.

Security Note

If your system is running the optional enhanced security, your login and password procedures may be different than the procedures documented in this book. See the *Security* document for more information.

1.1 Logging In

To use the operating system, your operating system must be installed and running and you must be logged in. Logging in identifies you as a valid system user and creates a work environment that belongs to you alone.

Before logging in, obtain your **username** and **password** from the system administrator. A username (typically your surname or initials) identifies you as an authorized user. A password (a group of characters that is easy for you to remember but difficult for others to guess) verifies your identity.

Think of your username and password as electronic keys that give you access to the system. When you enter your username and password during the login process, you identify yourself as an authorized user.

Your password is an important part of system security because it prevents unauthorized use of your data. For more information on passwords, see Section 1.5.1.

The first step in the login process is to display the login prompt. When your system is running and your workstation is on, the following login prompt appears on your screen:

```
login:
```

On some systems, you may have to press the Return key a few times to display the login prompt.

Your system's login prompt screen may be somewhat different. For example, in addition to the login prompt, the screen may display the system name and the version number of the operating system.

To log in, perform the following steps:

1. Enter your username at the login prompt. If you make a mistake, use the Delete key or Backspace key to correct it.

For example, if your username is larry, enter:

```
login: larry
```

The password prompt appears:

```
login: larry  
Password:
```

2. Enter your password. For security reasons, the password does not display on the screen when you type it.

If you think you made a mistake while typing your password, press the Return key. If your password is incorrect, the system displays a message and prompts you to enter your username and password again.

After you enter your username and password correctly, the system displays the shell prompt, usually a dollar sign (\$) prompt or a percent sign (%) prompt. Your system's shell prompt may be different.

Note

In this book, the shell prompt display is a dollar sign (\$).

The shell prompt display tells you that your login is successful, and that the system is ready to go to work for you. The shell prompt is your signal that the shell is running. The **shell** is a program that interprets all commands you enter, runs the programs you have asked for, and sends the results to your screen. For more information about commands and the shell prompt, see Section 1.3 and Chapter 7.

When you first log in, you are automatically placed in your **login** directory. See Chapter 2 for information about your login directory.

If your system does not display the shell prompt, you are not logged in. You may, for example, have entered your username or your password incorrectly. Try to log in again. If you still cannot log in, see your system administrator.

Note

Your system may not require you to have a password, or you may have been assigned a password that is common to all new users. To ensure security in these cases, set your own password. For information on how to create or change a password, see Section 1.5.

Many systems display a welcome message and announcements whenever users log in. For example, the following is a typical login screen (your screen may vary):

```
Welcome to the Operating System
Fri Dec 7 09:48:25 EDT 19nn
You have mail.
$
```

The preceding announcement contains the following pieces of information:

- A greeting
- The date and time of your last login

Note this information whenever you log in, and tell your system administrator if you have not logged in at the time specified. A wrong date and time might indicate that someone has been breaking into your system.

- Whether you have mail messages waiting to be read

Briefly, **mail** is a program that allows you to send and receive electronic mail. The system displays the message `You have mail` when there are mail messages for you that are waiting to be read. If you have no mail messages, this line does not appear.

1.2 Logging Out

When you are ready to end your work session, log out of the system. Logging out leaves the operating system running for other users and also ensures that no one else can use your work environment.

To log out, perform the following steps:

1. Make sure that the shell prompt is displayed.
2. Press Ctrl/D. If Ctrl/D does not work, enter the `exit` command.

The system displays the login prompt. On some systems, a message may also be displayed.

At this point, you or another user may log in.

1.3 Using Commands

Operating system commands are programs that perform tasks on the operating system. The operating system has a large set of commands that are described in the remaining chapters of this book and in the related reference pages.

Entering a command is an interactive process. When you enter a command, the shell interprets that command, and then gives an appropriate response – that is, the system either runs the program or displays an error message.

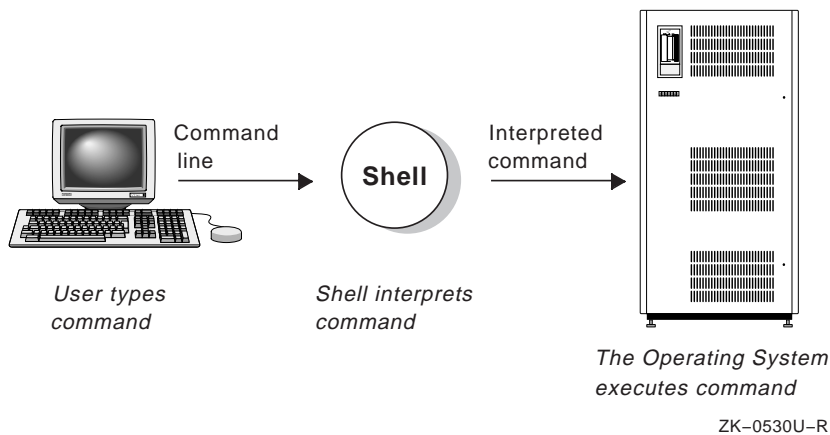
A shell reads every command you enter and directs the operating system to do what is requested. Therefore, the shell is a command interpreter.

The shell acts as a command interpreter in the following way:

1. The shell displays a shell prompt and waits for you to enter a command.
2. You enter a command, the shell analyzes it, and locates the requested program.
3. The shell asks the system to run the program, or it returns an error message.
4. When the program completes execution, control returns to the shell, which again displays the prompt.

Figure 1-1 shows the relationship between the user, the shell, and the operating system. The shell interacts with both the user to interpret commands and with the operating system to request command execution.

Figure 1-1: Shell Interaction with the User and the Operating System



The operating system supports three different shells: the Korn, C, and Bourne shells. Your system administrator determines which shell is active when you log in for the first time. For more information about shells, see Chapter 7.

When using the operating system, enter commands at the shell prompt on the command line. For example, to display today's date and time, enter:

```
$ date
```

If you make a mistake while typing a command, use the Delete key or Backspace key to erase the incorrect characters and then retype them.

An argument is a string of characters that follows a command name. An argument specifies the data the command uses to complete its action. For example, the `man` command gives you information about operating system commands. To display complete information about the `date` command, enter:

```
$ man date
```

Commands can have options that modify the way a command works. These options are called flags and immediately follow the command name. Most commands have several flags. If you use flags with a command, arguments follow the flags on the command line.

For example, suppose that you use the `-f` flag with the `man` command. This flag displays a one-line description of a specified command. To display a one-line description of the `date` command, you would enter:

```
$ man -f date
```

While a command is running, the system does not display the shell prompt because the control passes to the program you are running. When the command completes its action, the system displays the shell prompt again, indicating that you can enter another command.

In addition to using the commands provided with the system, you can also create your own personalized commands. Refer to Section 7.10.1 for information about creating these special commands.

1.4 Stopping Command Execution

If you enter a command and then decide that you do not want it to complete executing, enter `Ctrl/C`. The command call stops executing, and the system displays the shell prompt. You can now enter another command.

Depending upon the command, partial completion of the command may have varied results (referred to as an unknown state). To see the result of stopping a command during execution, enter `Ctrl/C` after executing commands such as `ls -l` to list files in a directory or `cat filename` to view a file on the screen.

1.5 Setting Your Password

Your username is public information and generally does not change. Your password, on the other hand, is private.

In most instances, when your system account is established, the system administrator assigns a password that is common to new users. After getting familiar with the system, select your own password to protect your account from unauthorized access. In addition, change your password periodically to protect your data from unauthorized access.

To set your password, use the `passwd` command. If your account does not have a password, use the `passwd` command to set one. For information on `passwd` procedures, see Section 1.5.2.

1.5.1 Password Guidelines

The following guidelines are useful in selecting a password:

- *Do not* choose a word found in a dictionary.
- *Do not* use personal information as your password, or as a substring of it, such as your username, names (yours, your family's, your company's), initials, or the make or model of your car.
- *Do not* use the default password you received with your account.
- *Do not* use old passwords or the same prefix or suffix you used in previous passwords. This rule also applies to any passwords you may have used in previous jobs.
- *Do not* choose a password that is easy to guess (includes all of the above options) even if you reverse their spelling. Choose a password that is hard to guess, not hard to remember.
- *Do not* choose passwords shorter than six characters in length. Your password can be up to eight characters long. (Password length is measured in bytes, rather than characters, but we can regard these terms as the same, for now.)
- *Do not* write your password on paper or place it in a file on the system.
- If possible, use a mixture of uppercase and lowercase letters in your password. You also should include any combination of numbers, punctuation marks, or underscores (`_`) in your password.
- Change your password frequently, especially if you think it might have been compromised.

On most systems, you can change your password as often or seldom as you like. However, to protect system security, your system administrator may set limits on how often you should change your password, the length of time your password remains valid, or the nature of changes you can make. Some typical password restrictions could be the following:

- Character restrictions
 - Minimum number of alphabetic characters
 - Minimum number of other characters, such as punctuation or numbers
 - Minimum number of characters in a new password that must be different from the old password
 - Maximum number of consecutive duplicate characters allowed in a password
- Time restrictions
 - Maximum number of weeks before your password expires

- Number of weeks before you can change a password

See your system administrator for more information about password restrictions.

1.5.2 Password Procedures

To set or change your password, follow these steps:

1. Enter the `passwd` command:

```
$ passwd
```

The system displays the following message (identifying you as the user) and prompts you for your old password:

```
Changing password for username  
Old password:
```

If you do not have an old password, the system does not display this prompt. Go to step 3.

2. Enter your old password. For security reasons, the system does not display your password as you enter it.

After the system verifies your old password, it is ready to accept your new password, and displays the following prompt:

```
New password:
```

3. Enter your new password at the prompt. Remember that your new password entry does not appear on the screen.

Finally, to verify the new password (since you cannot see it as you enter), the system prompts you to enter the new password again:

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

4. Enter your new password again. As before, the new password entry does not appear on the screen. When the shell prompt returns to the screen, your new password is in effect.

If you change your password and the new password does not conform to password regulations, you receive a message stating the specific problem and the restrictions in effect for the system.

Note

Try to remember your password because you cannot log in to the system without it. If you forget your password, see your system administrator.

1.6 Getting Help

Most operating system commands needed for your work are described in this book. If you want to learn more about these and other commands, see the reference pages. The reference pages are provided in three formats: online (see Section 1.6.1), Bookreader (see Chapter 5 in the *Installation Guide* for more information) and optionally in hard copy (see Related Documents in the About This Book section of this document). When the hard copy documents and Bookreader are unavailable, you can quickly access online command documentation by using the following commands:

- The `man` command displays online reference pages.
- The `apropos` command displays a one line summary of each command pertaining to a specified subject.

The following sections describe these features.

1.6.1 Displaying and Printing Online Reference Pages (`man`)

Online reference pages contain information about commands. To view a reference page online, use the `man` command. For example, to view the reference page for the `date` command, enter the following (your screen display may vary):

```
$ man date
date(1)                                     date(1)

NAME
  date - Displays or sets the date

SYNOPSIS

  Without Superuser Authority
  date [-u] [+field_descriptor ...]

  With Superuser Authority
  date [-nu] [MMddhhmm.ssyy | alternate_date_format] [+field_descriptor ...]

  The date command writes the current date and time to standard output.

DESCRIPTION

  The date command writes the current date and time to standard output if
  --More--(7%)
```

The symbol `--More-- (7%)` at the bottom of the page indicates that 7% of the reference page is currently displayed. At this point, you can press the Space bar to display the next screen of information, press the Return key to display one more line of information, or enter `q` to quit and return to the shell prompt.

Use the following command format to print a reference page:

```
man manpage | lpr -P printer_name
```

For example, to print the reference page for the `date` command on a specific printer, enter:

```
man date | lpr printer_name
```

The reference page for the `date` command is now queued for printing on *printer_name*. See Section 3.3 for more information about the `lpr` command.

To display a brief, one-line description of a command, use the `man -f` command. For example, to display a brief description of the `who` command, enter:

```
$ man -f who  
who (1) - Identifies users currently logged in  
$
```

For complete information on the `man` command and its options, you can display the reference page by entering the following:

```
$ man man
```

1.6.2 Locating Commands Using Descriptive Keywords

The `apropos` command and the `man -k` command are useful tools if you forget a command name.

Note

The `apropos` and the `man -k` commands require access to the `whatis` database. This database is available if your system manager loaded the default `whatis` database when the operating system was installed or created the database later using the `catman` command.

The `apropos` and `man -k` commands perform the same function. These commands allow you to enter a command description. The commands then list all the commands that fit that description.

As shown in the example, if a command description contains more than one word, the words must be enclosed in double quotes (" "). If the

command description contains only one word, it is not necessary to enclose the descriptive word in double quotes.

Assume that you cannot remember the name of the command that displays who is logged on to the system. To display the names and descriptions of all commands that have something to do with displaying users who are logged in, enter one of the following:

```
$ apropos "logged in"
```

or

```
$ man -k "logged in"
```

The system displays:

```
rwwho (1c) - who is logged in on local machines
users (1) - print names of users who are logged in
w (1) - display who is logged in and what they are doing
who (1) - identifies users currently logged in
```

Note

The numbers enclosed in parentheses refer to the section numbers of the reference pages.

After using the `apropos` or `man -k` commands, you now know that several commands: `rwwho`, `users`, `w`, and `who` can be used to display the users who are logged into the system.

Overview of Files and Directories **2**

This chapter provides an introduction to files, file systems, and text editors. A **file** is a collection of data stored together in the computer. Typical files contain memos, reports, correspondence, programs, or other data. A **file system** is the useful arrangement of files into **directories**.

A **text editor** is a program that allows you to create new files and modify existing ones.

After completing this chapter, you will be able to:

- Create files with the `vi` text editor. These files will be useful for working through the examples later in this book.
- Understand the file system components and concepts.

This knowledge can help you design a file system that is appropriate for the type of information you use and the way you work.

2.1 Overview of Text Editors

An editor is a program that allows you to create and change files containing text, programs, or other data. An editor does not provide the formatting and printing features of a word processor.

With a text editor, you can:

- Create, read, and write files
- Display and search for data
- Add, replace, and remove data
- Move and copy data
- Run operating system commands

Your editing takes place in an edit buffer that you can save or discard.

The `vi` and `ed` text editing programs are available on the operating system. Each editor has its own methods of displaying text as well as its own set of subcommands and rules.

For information about `vi`, read the next section and Appendix A. For information about `ed`, see Appendix B.

Your system may contain additional editors; see your system administrator for details.

2.2 Creating Sample Files with the vi Text Editor

This section shows how to create three files with the vi text editor.

The goal of this section is to have you create, using a minimal set of commands, files that can be used for working through the examples later in this book. For more information about vi, see Appendix A and the vi(1) reference page.

Note

If you are familiar with a different editing program, you can use that program to create the three sample files described in this section. If you have already created three files with an editing program, you can use those files by substituting their names for the filenames used in the examples.

When following the steps that are used to create the sample files, only enter the text that is shown in **boldface** characters. System prompts and output are shown in a different typeface, like this.

To create three sample files, follow these steps:

1. Start the vi program by typing vi and the name of a new file at the shell prompt. Press the Return key:

```
$ vi file1 Return :
```

This is a new file, so the system responds by putting your cursor at the top of a screen:

```
~  
~  
~  
~  
~  
~  
~  
"file1" [New file]
```

Note the blank lines on your screen that begin with a tilde (~). These tildes indicate the lines that contain no text. Because you have not entered any text, all lines begin with a tilde.

2. To specify that you want to insert text to the new file, type the lowercase letter i. The system does not display the i that you enter.

Enter the following sample text, pressing the Return key after each line. To correct mistakes before moving to the next line, press the Delete key or the Backspace key to move backward over the mistake. Retype the text correctly.

```
You start the vi program by entering Return
the vi command optionally followed by the name Return
of a new or existing file. Escape
~
~
~
~
~
~
"file1" [New file]
```

3. To indicate that you have finished your current work, press the Escape key and type a colon (:).

Note

Depending upon how your terminal or workstation is set up, the Escape key may be programmed to perform a different function. It is possible that one of the function keys on your keyboard may have been set up to perform the "escape" function. See your system administrator if your Escape key does not operate properly.

The colon is displayed as a prompt at the bottom of the screen as follows:

```
You start the vi program by entering
the vi command optionally followed by the name
of a new or existing file.
~
~
~
~
~
~
~
:
```

4. Next, enter a lowercase letter `w`. Entering the letter `w` indicates to the system that you want to write, or save, a copy of the new file in your current, user directory (see Chapter 4 for an explanation about your current directory).

Your screen will look like this:

```
You start the vi program by entering
the vi command optionally followed by the name
of a new or existing file.
```

```
~
~
~
~
~
~
```

```
"file1" [New file] 3 lines, 111 characters
```

The system displays the name of the new file as well as the number of lines and characters it contains.

The system is still in the vi text editor so you can create two more sample files. The process is the same as the one you used to create file1, but the text you enter will be different.

5. To create the second file, file2, type a colon (:). The colon is displayed as a prompt at the bottom the screen. Enter vi file2 to create your second sample file.

The system responds with a screen that looks like this:

```
~
~
~
~
~
~
~
```

```
"file2" No such file or directory
```

The message file2 No such file or directory indicates that file2 is a new file.

6. Indicate that you want to insert text to the new file by typing the lowercase letter i. Enter the following sample text:

```
If you have created a new file, you will find
that it is easy to add text.
```

7. Press the Escape key, type a colon (:), and enter the lowercase letter w to write, or save, the file in your current directory.

Your screen will look like this:

```
If you have created a new file, you will find
that it is easy to add text.
```

```
~
~
~
~
~
~
```

```
"file2" [New file] 2 lines, 75 characters
```

8. To create the third file, follow the instructions in step 5. However, name the file `file3`, and enter the following sample text:

```
You will find that vi is a useful
editor that has many features.
```

9. Then, press the Escape key, type a colon (:), and enter the `wq` command. The `wq` command writes the file, quits (or exits) the editor, and returns you to the shell prompt.

2.3 Understanding Files, Directories, and Pathnames

A **file** is a collection of data stored in a computer. A file stored in a computer is like a document stored in a filing cabinet because you can retrieve it, open it, process it, close it, and store it as a unit. Every computer file has a **filename** that both users and the system use to refer to the file.

A **file system** is the arrangement of files into a useful pattern. Any time you organize information, you create something like a computer file system. For example, the structure of a manual file system (file cabinets, file drawers, file folders, and documents) resembles the structure of a computer file system. (The software that manages the file storage is also known as the file system, but that usage of the term does not occur in this chapter.)

Once you have organized your file system (manual or computer), you can find a particular piece of information quickly because you understand the structure of the system. To understand the file system, you should first become familiar with the following three concepts:

- Files and filenames
- Directories and subdirectories
- Tree structures and pathnames

2.3.1 Files and Filenames

A file can contain the text of a document, a computer program, records for a general ledger, the numerical or statistical output of a computer program, or other data.

A file name can contain any character except the following because these characters have special meaning to the shell:

- slash (/)
- backslash (\)
- ampersand (&)
- left- and right-angle brackets (< and >)
- question mark (?)
- dollar sign (\$)
- left bracket ([)
- asterisk (*) or
- vertical bar or pipe symbol (|)

You may use a period or dot (.) in the middle of a filename, but never at the beginning of the filename unless you want the file to be "hidden" when doing a simple listing of files. For information about characters with special meanings to your shell, refer to the sections about metacharacters in Chapter 8. For information about listing hidden files, see Section 3.1.3.

Note

Unlike some operating systems, this operating system distinguishes between uppercase and lowercase letters in filenames (that is, it is case sensitive). For example, the following three filenames represent three distinct files: `filea`, `Filea`, and `FILEA`.

Use filenames that reflect the actual contents of your files. For example, a filename such as `memo.advt` might indicate that the file contains a memo about advertising. On the other hand, filenames such as `filea`, `fileb`, or `filec` tell you nothing about the contents of that file.

It is also a good idea to use a consistent pattern to name related files. For example, suppose you have an advertising report that is divided into chapters, with each chapter contained in a separate file. You might name these files in the following way:

```
chap1.advt
chap2.advt
chap3.advt
```

The maximum length of a filename depends upon the file system used on your operating system. For example, your file system may allow a maximum filename length of 255 characters (the default), or it may allow a maximum filename length of only 14 characters. Because knowing the maximum filename length is important to name files with meaningful file names, see your system administrator for details.

2.3.2 Directories and Subdirectories

You can organize your files into groups and subgroups that resemble the cabinets, drawers, and folders in a manual file system. These groups are called **directories**, and the subgroups are called **subdirectories**. A well-organized system of directories and subdirectories lets you retrieve and manipulate the data in your files quickly.

Directories differ from files in two significant ways:

- Directories are organizational tools; files are storage places for data.
- Directories contain the names of files, other directories, or both.

When you first log in, the system automatically places you in your **login** directory. This directory was created for you when your computer account was established. However, a file system in which all files are arranged under your login directory is not necessarily the most efficient method to organize your files.

As you work with the system, you may want to set up additional directories and subdirectories so you can organize your files into useful groups. For example, assume that you work for the Sales department and are responsible for four lines of automobiles. You may want to create a subdirectory under your login directory for each automobile line. Each subdirectory can contain all memos, reports, and sales figures applicable for the automobile model.

Once your files are arranged into a directory structure that you find useful, you can move easily between directories. See Chapter 4 for information about creating directories and moving between them.

2.3.3 Displaying the Name of Your Current (Working) Directory (pwd)

The directory in which you are working at any given time is your **current**, or working directory.

Whenever you are uncertain about the directory in which you are working or where that directory exists in the file system, enter the `pwd` (print working directory) command as follows:

```
$ pwd
```

The system displays the name of your current directory in the format:

```
/usr/msg
```

This information indicates that you are currently working in a directory named `msg` that is located under the `usr` directory.

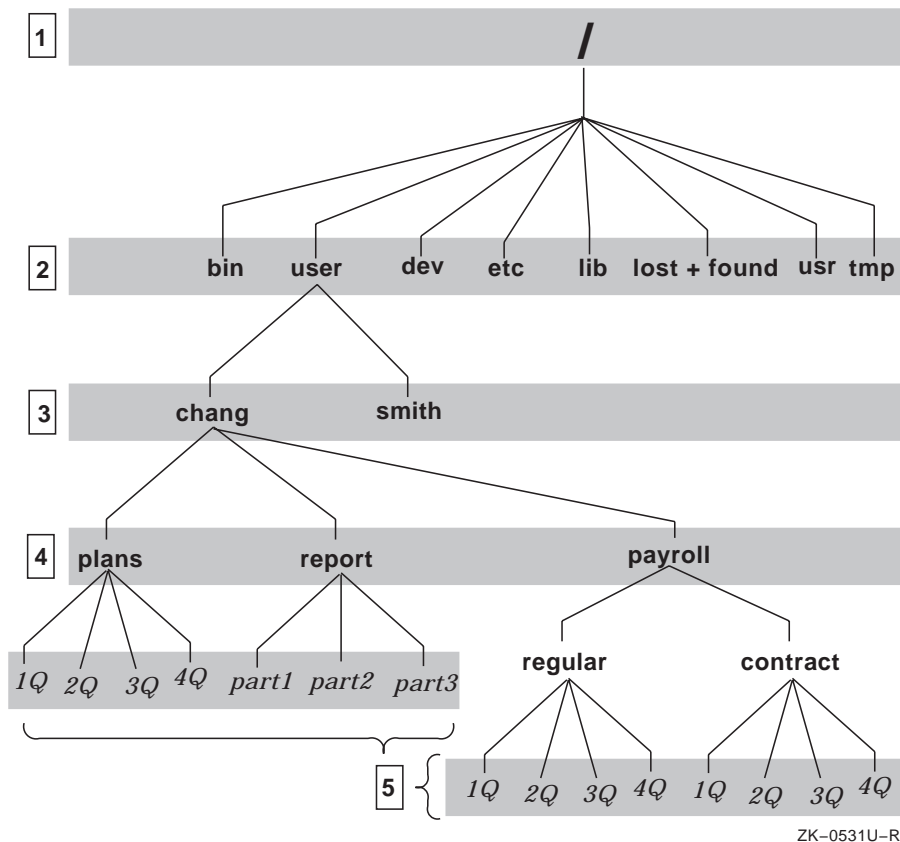
The `/usr/msg` notation is known as the **pathname** of your working directory. See the following section for information about pathnames.

2.3.4 The Tree-Structure File System and Pathnames

The files and directories in the file system are arranged hierarchically in a structure that resembles an upside-down tree with the roots at the top and the branches at the bottom. This arrangement is called a **tree** structure. You can find more detailed information about the directory structure in the `hier(5)` reference page.

Figure 2-1 shows a typical file system arranged in a tree structure. The names of directories are printed in **bold**, and the names of files are printed in *italics*.

Figure 2-1: A Typical Digital UNIX File System



- 1 At the top of the file system shown in Figure 2-1 (that is, at the root of the inverted tree structure) is a directory called the `root` directory. The symbol that represents this first major division of the file system is a slash (/).
- 2 At the next level down from the root of the file system are eight directories, each with its own system of subdirectories and files. Figure 2-1, however, shows only the subdirectories under the directory named `user`. These are the login directories for the users of this system.
- 3 The third level down the tree structure contains the login directories for two of the system's users, `smith` and `chang`. It is in these directories that `smith` and `chang` begin their work after logging in.
- 4 The fourth level of the figure shows three directories under the `chang` login directory: `plans`, `report`, and `payroll`.

- 5 The fifth level of the tree structure contains both files and subdirectories. The `plans` directory contains four files, one for each quarter. The `report` directory contains three files comprising the three parts of a report. Also on the fifth level are two subdirectories, `regular` and `contract`, which further organizes the information in the `payroll` directory.

A higher level directory is frequently called a **parent** directory. For example, in Figure 2-1, the directories `plans`, `report`, and `payroll` all have `chang` as their parent directory.

A **pathname** specifies the location of a directory or a file within the file system. For example, when you want to change from working on File A in Directory X to File B in Directory Y, you enter the pathname to File B. The operating system then uses this pathname to search through the file system until it locates File B.

A pathname consists of a sequence of directory names separated by slashes (/) that ends with a directory name or a filename. The first element in a pathname specifies where the system is to begin searching, and the final element specifies the target of the search. The following pathname is based on Figure 2-1:

```
/user/chang/report/part3
```

The first slash (/) represents the root directory and indicates the starting place for the search. The remainder of the pathname indicates that the search is to go to the `user` directory, then to the `chang` directory, next to the `report` directory, and finally to the `part3` file.

Whether you are changing your current directory, sending data to a file, or copying or moving a file from one place in your file system to another, you use pathnames to indicate the objects you want to manipulate.

A pathname that starts with a slash (/) (the symbol representing the root directory) is called a **full pathname** or an **absolute pathname**. You can also think of a full pathname as the complete name of a file or a directory. Regardless of where you are working in the file system, you can always find a file or a directory by specifying its full pathname.

The file system also lets you use **relative pathnames**. Relative pathnames do not begin with the / that represents the root directory because they are relative to the current directory.

You can specify a relative pathname in one of several ways:

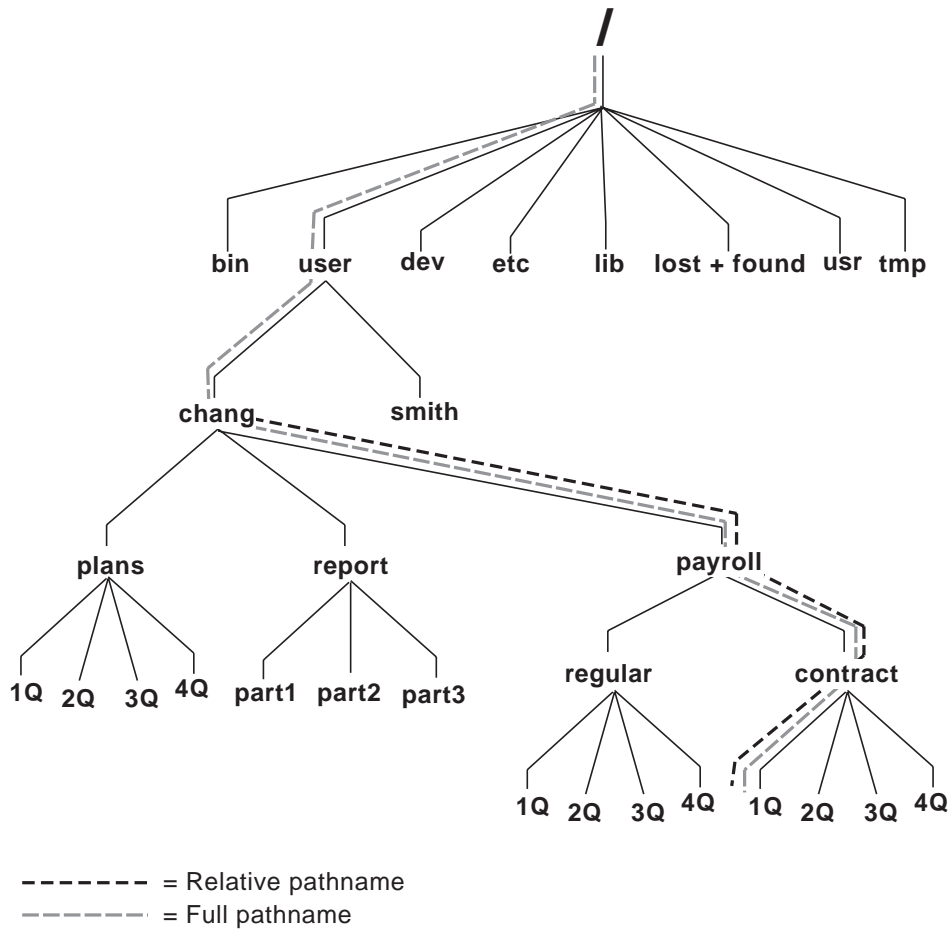
- As the name of a file in the current directory.
- As a pathname that begins with the name of a directory one level below your current directory.

- As a pathname that begins with `..` (dot dot, the relative pathname for the parent directory).
- As a pathname that begins with `.` (dot, which refers to the current directory). This relative pathname notation is useful when you want to run your own version of an operating system command in the current directory (for example `./ls`).

Every directory contains at least two entries: `..` (dot dot), and `.` (dot, which refers to the current directory).

In Figure 2-2, for example, if your current directory is `chang`, the relative pathname for the file `1Q` in the `contract` directory is `payroll/contract/1Q`. By comparing this relative pathname with the full pathname for the same file, `/user/chang/payroll/contract/1Q`, you can see that using relative pathnames means less typing and more convenience.

Figure 2-2: Relative and Full Pathnames



ZK-0532U-R

In the C shell and the Korn shell, you may also use a tilde (~) at the beginning of relative pathnames. The tilde character specifies a user's login (home) directory.

For example, to specify your own login directory, use the tilde alone. To specify the login directory of user chang, specify ~chang.

For more information on using relative pathnames, see Chapter 4.

Note

If there are other users on your system, you may or may not be able to get to their files and directories, depending upon the permissions set for them. For more information about file and directory permissions, see Chapter 5. In addition, your system may contain enhanced security features that may affect access to files and directories. If so, see your system administrator for details.

2.4 Specifying Files with Pattern Matching

Commands often take filenames as arguments. To use several different filenames as arguments to a command, you can type out the full name of each file, as the following example shows:

```
$ ls file1 file2 file3
```

However, if the filenames have a common pattern (in this example, the `file` prefix), the shell can match that pattern, generate a list of those names, and automatically pass them to the command as arguments.

The asterisk (`*`), sometimes referred to as a wildcard, matches any string of characters. In the following example, the `ls` command finds the name of every text file in the current directory that includes the `file` prefix:

```
$ ls file*
```

The `file*` matches any filename that begins with `file` and ends with any other character string. The shell passes every filename that matches this pattern as an argument for the `ls` command.

Thus, you do not have to enter (or even remember) the full name of each file in order to use it as an argument. Both commands (`ls` with all filenames typed out and `ls file*`) do the same thing – they pass all files with the `file` prefix in the directory as arguments to the `ls` command.

There is one exception to the general rules for pattern matching. When the first character of a filename is a period, you must match the period explicitly. For example, `ls *` displays the names of all files in the current directory that do not begin with a period. The command `ls -a` displays all filenames that begin with a period.

This restriction prevents the shell from automatically matching the relative directory names. These are `.` (called dot, standing for the current directory) and `..` (called dot dot, standing for the parent directory). For more information on relative directory names, see Chapter 4.

If a pattern does not match any filenames, the shell displays a message informing you that no match has been found.

In addition to the asterisk (*), operating system shells provide other ways to match character patterns. The following list summarizes all pattern-matching characters and provides examples.

Character	Action
*	Matches any string, including the null string. For example, <code>th*</code> matches <code>th</code> , <code>theodore</code> , and <code>theresa</code> .
?	Matches any single character. For example, <code>304?b</code> matches <code>304Tb</code> , <code>3045b</code> , <code>304Bb</code> , or any other string that begins with <code>304</code> , ends with <code>b</code> , and has one character in between.
[...]	Matches any one of the enclosed characters. For example, <code>[AGX]*</code> matches all filenames in the current directory that begin with <code>A</code> , <code>G</code> , or <code>X</code> .
[.-]	Matches any character that falls within the specified range, as defined by the current locale. For more information on locale, see Appendix C. For example, <code>[T-W]*</code> matches all filenames in the current directory that begin with <code>T</code> , <code>U</code> , <code>V</code> , or <code>W</code> .
[!...]	Matches any single character except one of those enclosed. For example, <code>[!abyz]*</code> matches all filenames in the current directory that begin with any character except <code>a</code> , <code>b</code> , <code>y</code> , or <code>z</code> . This pattern matching is available only in the Bourne and Korn shells.

Because this operating system is an internationalized operating system, it provides the following additional pattern-matching features:

Character	Action
<code>[:class:]</code>	A character class name enclosed in bracket-colon delimiters matches any of the set of characters in the named class. The supported classes are <code>alpha</code> , <code>upper</code> , <code>lower</code> , <code>digit</code> , <code>alnum</code> , <code>xdigit</code> , <code>space</code> , <code>print</code> , <code>punct</code> , <code>graph</code> , and <code>cntrl</code> . For example, the <code>alpha</code> character class name specifies that you want to match any alphabetic character

(uppercase and lowercase) as defined by the current locale. If you are running an American-based locale, `alpha` matches any character in the alphabet (A-Z, a-z).

`[[=char=]*]`

A character enclosed in bracket-equal delimiters matches any equivalence class character.

An equivalence class is a set of collating elements that all sort to the same primary location. It is generally designed to deal with primary-secondary sorting; that is, for languages such as French that define groups of characters as sorting to the same primary location, and then having a tie-breaking, secondary sort.

For more information on internationalized pattern-matching characters, see the `grep(1)` reference page. For more information on the operating system's internationalization features, see Appendix C.

Managing Files **3**

This chapter describes how to manage files on your system. After completing this chapter, you will be able to:

- List files
- Display and print files
- Link files
- Copy, rename, and move files
- Compare and sort files
- Remove files from the system
- Determine file type

To learn about managing files, follow the examples in this chapter. Do each example in order so that the information on your screen is consistent with the information in this book.

Before you can work through the examples, you must be logged in and your login directory must contain the following three files created in Chapter 2: `file1`, `file2`, and `file3`. To produce a listing of the files in your login directory, enter the `ls` command, which is explained in the following section. If you are using files with different names, make the appropriate substitutions as you work through the examples.

In the following examples, when you are asked to return to your login directory, enter the `cd` (change directory) command as follows:

```
$ cd  
$
```

In the preceding example, the dollar sign (\$) represents the shell prompt. Your shell prompt may vary.

In addition, before working on the examples in this chapter, create a subdirectory called `project` in your login directory. To do so, enter the following `mkdir` (make directory) command from your login directory:

```
$ mkdir project
$
```

For more information on the `cd` and `mkdir` commands, see Section 4.2 and Section 4.1 respectively.

3.1 Listing Files (`ls`)

You can display a listing of the contents of one or more directories with the `ls` (list directory) command. This command produces a list of the files and subdirectories (if any) in your current directory. You can also display other types of information, such as the contents of directories other than your current directory.

The format of the `ls` command is:

ls

The `ls` command has a number of options, called **flags** that enable you to display different types of information about the contents of a directory. Refer to Section 3.1.3 for information about these flags.

3.1.1 Listing Contents of the Current Directory

To list the contents of your current directory, enter:

```
$ ls
```

Used without flags in this format, the `ls` command lists the names of the files and directories in your current directory:

```
$ ls
file1      file2      file3      project
$
```

You may also list portions of your current directory's contents by using the command format:

ls *filename*

The *filename* entry can be the name of the file or a list of filenames separated by spaces. You may also use pattern-matching characters to specify files. See Chapter 2 for information on pattern matching.

For example, to list the files whose names begin with the characters `file`, you would enter the following command:

```
$ ls file*
file1 file2 file3
$
```

3.1.2 Listing Contents of Other Directories

To display a listing of the contents of a directory other than your current directory, use the following format:

ls *dirname*

The *dirname* entry is the pathname of the directory whose contents you want to display.

In the following example, the current directory is your login directory, and you want to display the `/users` directory. Your system may contain another directory with a name similar to the `/users` directory. The name of the `/users` directory is preceded by a slash (`/`), which indicates that the system should begin searching from the root directory.

```
$ ls /users
amy      beth      chang     george    jerry     larry
mark     monique  ron
$
```

The `ls` command lists directory and filenames in collated order as determined by the current locale. For more information about locales (as used with internationalization), see Appendix C.

3.1.3 Flags Used with the ls Command

In its simplest form, the `ls` command displays only the names of files and directories contained in the specified directory. However, `ls` has several flags that provide additional information about the listed items or change the way in which the system displays the listing.

When you want to include flags with the `ls` command, use the following format:

ls *-flagname(s)*

The *-flagname(s)* entry specifies one or more flags (options) that you are using with the command. For example, the `-l` flag produces a long listing of the directory contents. Note also that all `ls` flags are preceded by the dash character (`-`).

If you want to use multiple flags with the command, enter the flag names together in one string:

```
$ ls -lta
```

Table 3-1 lists some of the most useful `ls` command flags.

Table 3-1: The `ls` Command Options

Flag	Action
<code>-l</code>	Lists in long format. An <code>-l</code> listing provides the type, permissions, number of links, owner, group, size, and time of last modification for each file or directory listed.
<code>-t</code>	Sorts the files and directories by the time they were last modified (latest first), rather than collated by name.
<code>-r</code>	Reverses the order of the sort to get reverse collated order (<code>ls -r</code>), or reverse time order (<code>ls -tr</code>).
<code>-a</code>	Lists all entries including "hidden files". Without this flag, the <code>ls</code> command does not list the names of entries that begin with a dot (<code>.</code>), such as <code>.profile</code> , <code>.login</code> , and relative pathnames.

The following example shows a long (`-l`) listing of a current directory. The name `larry` shows the owner of the files. Your username and group name will replace `larry` and `system` on the screen.

```
$ ls -l
total 4
-rw-r--r-- 1 larry system 101 Jun  5 10:03 file1
-rw-r--r-- 1 larry system  75 Jun  5 10:03 file2
-rw-r--r-- 1 larry system  65 Jun  5 10:06 file3
drwxr-xr-x 2 larry system  32 Jun  5 10:07 project
$
```

Table 3-2 explains the information displayed on your screen after you enter the `ls -l` command.

Table 3-2: The ls -l Command Information

Field	Information
total 4	Number of 512-byte blocks taken up by files in this directory.
drwxr-xr-x	File type and permissions set for each file or directory. The first character in this field indicates file type: – (dash) for ordinary files b for block-special files c for character-special files d for directories l for symbolic links p for pipe-special files (first in, first out) s for local sockets The remaining characters indicate what read (r), write (w), and execute (x) permissions are set for the owner, group, and others. In addition, other permission information may also be displayed. ^a
1	Number of links to each file. ^b
larry	Username of the file's owner.
system	Group to which the file belongs.
101	Number of bytes in the file.
Jun 5 10:03	Date and time the file was created or last modified in the format defined by your current locale.
file1	Name of the file or directory.

Table notes:

- a. For more information on permissions, see Chapter 5.
- b. For an explanation of file links, see Section 3.4.

There are other `ls` command flags that you may find useful as you gain experience with the operating system. For detailed information about the `ls` command flags, see the `ls(1)` reference page.

3.2 Displaying Files

You can view any text file stored on your system with a text editor. However, if you want to just look at a file without making any changes, you may view it (with or without screen formatting) using a variety of operating system commands. The following sections describe these commands.

3.2.1 Displaying Files Without Formatting (pg, more, cat)

The following commands display a file just as it is, without adding any special characteristics that govern the appearance of the contents:

- `pg`
- `cat`
- `more`

For information on displaying files with formatting, see Section 3.2.2.

To display a file without formatting, the general format is:

command filename

The *command* entry is one of the following command names: `pg`, `more`, or `cat`. The *filename* entry can be the name of one file, or a series of filenames separated by spaces. You may also use pattern-matching characters to specify your files. See Chapter 2 for information on using pattern-matching characters.

The `pg` command allows you to view one or more files. In the following example, the `pg` command displays the contents of `file1` in your login directory:

```
$ pg file1
```

```
You start the vi program by entering  
the command vi, optionally followed by the name  
of a new or existing file.
```

```
$
```

To view the contents of both `file1` and `file2`, enter both file names on the command line. When you display files that contain more lines than will fit on the screen, the `pg` command pauses as it displays each screen. To view the next screen of information in a file, press the Return key until you reach the end of the current file. When you press the Return key at the end of the current file, the start of the next file is displayed. The `pg` command always displays multiple files in the order in which you listed them on the command line. In the example shown below, (EOF): (end of file) means that you are at the end of the current file.

```
$ pg file1 file2You start the vi program by entering  
the command vi, optionally followed by the name  
of a new or existing file.
```

```
(EOF): Return
```

```
(Next file: file2) Return
```

```
If you have created a new file, you will find  
that it is easy to add text.
```

```
(EOF): Return
```

```
$
```

At the `Next file: filename` prompt, you can enter the `-n` option to go back to the previous file instead of displaying the next file.

When you display files that contain more lines than will fit on the screen, the `pg` command pauses as it displays each screen. To see the next screen of information in a file, press the Return key.

The `more` command also allows you to enter multiple file names on the command line and is very much like the `pg` command in the way that it handles long files. If the file(s) contains more lines than can fit on your screen, `more` pauses and displays a message telling you what percentage of the file you have viewed thus far. At this point, you can do one of the following:

- Press the Space bar to display the remainder of the file a page at a time
- Press the Return key to display one line at a time
- Type `q` to quit viewing the file(s)

The `cat` command also displays text. However, it is less useful for viewing long files because it does not paginate files. When viewing a file that is larger than one screen, the contents will display too quickly to be read. When this happens, press `Ctrl/S` to halt the display. You can then read the text. When you want to display the remainder of the file, press `Ctrl/Q`. Because `cat` is not useful for viewing long files, you may prefer using the `pg` or `more` command in these cases.

The `pg`, `more`, and `cat` commands all have additional options that you may find useful. For more information, refer to the reference pages for these commands.

3.2.2 Displaying Files with Formatting (`pr`)

Formatting is the process of controlling the way the contents of your files appear when you display or print them. The `pr` command displays a file in a simple but useful style.

Note

The `pr` command does not interpret any text formatting macros that may reside in your file. The `pr` command does not format files the same way as `nroff` or `troff`, for example.

To display a file with simple formatting, the format of the command is:

```
pr filename
```

The *filename* entry can be the name of the file, the relative pathname of the file, the full pathname of the file, or a list of filenames separated by

spaces. The format you use depends on where the file is located in relation to your current directory. You may also use pattern-matching characters to specify files. See Chapter 2 for information on pattern matching.

Used without any options, the `pr` command does the following:

- Divides the contents of the file into pages
- Puts the date, time, page number, and filename in a heading at the top of each page
- Leaves five blank lines at the end of the page

When you use the `pr` command to display a file, its contents may scroll off your screen too quickly for you to read them. When this happens, you can view the formatted file by using the `pr` command along with the `more` command. The `more` command instructs the system to pause at the end of each screenful of text.

For example, to display a long file called `report` so that it pauses when the screen is full, enter the following command:

```
$ pr report more
```

When the system pauses at the first screen of text, press the Space bar to display the next screen. The previous command uses the pipe symbol (`|`) to take the output from the `pr` command and use it as input to the `more` command. For more information on pipes, see Section 7.4.2.

Sometimes you may prefer to display a file in a more sophisticated format. You can use a number of flags in the command format to specify additional formatting features. Table 3-3 explains several of these flags.

Table 3-3: The `pr` Command Flags

Flag	Action
<code>+page</code>	Begins formatting on page number <i>page</i> . Otherwise, formatting begins on page 1. For example, the <code>pr +2 file1</code> command starts formatting <code>file1</code> on page 2.
<code>-column</code>	Formats page into columns. Otherwise, <code>pr</code> formats pages with one column. For example, the <code>pr -2 file1</code> command formats <code>file1</code> into two columns.

Table 3-3: (continued)

Flag	Action
-m	Formats all specified files at the same time, side-by-side, one per column. For example, the <code>pr -m file1 file2</code> command displays the contents of <code>file1</code> in the left column, and that of <code>file2</code> in the right column.
-d	Formats double-spaced output. Otherwise, output is single-spaced. For example, the <code>pr -d file1</code> command displays <code>file1</code> in double-spaced format.
-f	Uses a formfeed character to advance to a new page. (Otherwise, <code>pr</code> issues a sequence of linefeed characters.) Pauses before beginning the first page if the standard output is a terminal.
-F	Uses a formfeed character to advance to a new page. (Otherwise, issues a sequence of linefeed characters.) Does not pause before beginning the first page if the standard output is a terminal.
-wnum	Sets line width to <i>num</i> columns. Otherwise, line width is 72 columns. For example, the <code>pr -w40 file1</code> command sets the line length of <code>file1</code> to 40 columns.
-onum	Offsets (indents) each line by <i>num</i> column positions. Otherwise, offset is 0 (zero) column positions. For example, the <code>pr -o5 file1</code> command indents each line of <code>file1</code> five spaces.
-lnum	Sets page length to <i>num</i> lines. Otherwise, page length is 66 lines. For example, the <code>pr -l30 file1</code> command sets the page length of <code>file1</code> to 30 lines.
-h <i>string</i>	Uses the specified string of characters, rather than the filename, in the header (title) that is displayed at the top of every page. If <i>string</i> includes blanks or special characters, it must be enclosed in ' ' (single quotes). For example, the <code>pr -h 'My Novel' file1</code> command specifies "My Novel" as the title.
-t	Prevents <code>pr</code> from formatting headings and the blank lines at the end of each page. For example, the <code>pr -t file1</code> command specifies that <code>file1</code> be formatted without headings and blank lines at the end of each page.

Table 3-3: (continued)

Flag	Action
<code>-schar</code>	Separates columns with the character <i>char</i> rather than with blank spaces. You must enclose special characters in single quotes. For example, the <code>pr -s '*' file1</code> command specifies that asterisks separate columns.

You can use more than one flag at a time with the `pr` command. In the following example, you instruct `pr` to format `file1` with these characteristics:

- With double spacing (`d`)
- With the title "My Novel" rather than the name of the file

```
$ pr -dh 'My Novel' file1
```

For detailed information about `pr` and its flags, refer to the `pr(1)` reference page.

3.3 Printing Files (`lpr`, `lpq`, `lprm`)

Use the `lpr` command to send one or more files to the system printer. The `lpr` command actually places files in a **print queue**, which is a list of files waiting to be printed. Once the `lpr` command places your files in the queue, you can continue to do other work on your system while you wait for the files to print.

The general format of the `lpr` command is:

```
lpr filename
```

The *filename* entry can be the name of the file, the relative pathname of the file, the full pathname of the file, or a list of filenames separated by spaces. The format you use depends on where the file is located in relation to your current directory. You may also use pattern-matching characters to specify files. See Chapter 2 for information on pattern matching.

If your system has more than one printer, use the following format to specify where you want the file to print:

```
lpr -P printername filename
```

The `-P` flag indicates that you want to specify a printer. The *printername* entry is the name of a printer. Printers often have names such as `lp0`, `lp1`, and `lpn`. Ask your system administrator for the printer names at your facility.

If your system has more than one printer, one of them is the default printer. When you do not enter a specific *printername*, your print request goes to the default printer.

The following example shows how to use the `lpr` command to print one or more files on a printer named `lp0`:

```
$ lpr -Plp0 file1
$ lpr -Plp0 file2 file3
$
```

The first `lpr` command sends `file1` to the `lp0` printer and then displays the shell prompt: a dollar sign (`$`). The second `lpr` command sends `file2` and `file3` to the same print queue, and then displays the shell prompt before the files finish printing.

Several `lpr` command flags enable you to control the way in which your file prints. Following is the general format for using a flag with this command:

`lpr flag filename`

Table 3-4 explains some of the most useful `lpr` command flags.

Table 3-4: The `lpr` Command Flags

Flag	Action
<code>-#num</code>	Prints <i>num</i> copies of the file. Otherwise, <code>lpr</code> prints one copy. For example, the <code>lpr -#2 file1</code> command prints two copies of <code>file1</code> .
<code>-wnum</code>	Sets line width to <i>num</i> columns. Otherwise, line width is 72 columns. For example, the <code>lpr -w40 file1</code> command prints <code>file1</code> with lines that are 40 columns long.
<code>-inum</code>	Offsets (indents) each line by <i>num</i> space positions. Otherwise, offset is 8 spaces. For example, the <code>lpr -i5 file1</code> command prints <code>file1</code> with lines that are indented five spaces.
<code>-p</code>	Formats the file using <code>pr</code> as a filter.

Table 3-4: (continued)

Flag	Action
-T <i>'string'</i>	Uses the specified string of characters, rather than the filename, in the header used by <code>pr</code> . Requires the <code>-p</code> option. If the string includes blanks or special characters, it must be enclosed in <code>' '</code> (single quotes). For example, the <code>lpr -p -T 'My Novel' file1</code> command specifies "My Novel" as the title.
-m	Sends mail when the file completes printing. For example, the command <code>lpr -m file1</code> specifies that you want mail to be sent to you once <code>file1</code> prints.

Once you have entered the `lpr` command, your print request is entered into the print queue.

To see the position of the request in the print queue, use the `lpq` command. To look at the print queue, enter:

```
$ lpq
```

If your request has already been printed, or if there are no requests in the print queue, the system responds with the following message:

```
no entries
```

If there are entries in the print queue, the system lists them and indicates which request is currently being printed. Following is a typical listing of print queue entries (your listing will vary):

Rank	Owner	Job	Files	Total Size
active	marilyn	489	report	8470 bytes
1st	sue	135	letter	5444 bytes
2nd	juan	360	(standard input)	969 bytes
3rd	larry	490	travel	1492 bytes

The `lpq` command displays the following for each print queue entry:

- Its priority
- Its owner
- Its job number
- Name of the file
- Size of the file in bytes

For example, Marilyn's report (job number 489) is currently being printed, and the requests of Sue, Juan, and Larry are pending.

When you print files, the position of the request in the queue as well as its size may help you estimate when your request may be finished. Generally, the higher the priority number in the queue and the larger the print request, the more time it will take.

If your system has more than one printer, use the following format to specify which print queue you want to see:

lpq -P *printername filename*

The `-P` flag indicates that you want to specify a print queue. The *printername* entry is the name of a particular printer. Use the `lpstat -s` command to learn the names of all the printers.

If you decide not to print your request, you can delete it from the print queue by using the `lprm` command. The general format of the `lprm` command is the following:

lprm *jobnumber*

The *jobnumber* entry specifies the job number that the system has assigned to your print request. You can see the job number by entering the `lpq` command.

For example, if Larry wants to cancel his print request, he can enter:

```
$ lprm 490
$
```

The `travel` file will be removed from the print queue.

3.4 Linking Files (ln)

A `link` is a connection between a filename and the file itself. Usually, a file has one link – a connection to its original filename. However, you can use the `ln` (`link`) command to connect a file to more than one filename at the same time.

Links are convenient whenever you need to work with the same data in more than one place. For example, suppose you have a file containing assembly-line production statistics. You use the data in this file in two different documents – in a monthly report prepared for management, and in a monthly synopsis prepared for the line workers.

You can link the statistics file to two different filenames, for example, `mgmt.stat` and `line.stat`, and place these filenames in two different directories. In this way, you save storage space because you have only one copy of the file. More importantly, you do not have to update multiple files. Because `mgmt.stat` and `line.stat` are linked, editing one automatically updates the other, and both filenames always refer to the same data.

3.4.1 Hard Links and Soft Links

There are two kinds of links available for your use: hard links and soft (symbolic) links.

- Hard links allow you to link only files in the same file system. When you create a hard link, you are providing another name for the same file. All the hard link names for a file, including the original name, are on equal footing. It is incorrect to think of one filename as the real name, and another as only a link.
- Soft links or symbolic links allow you to link both files and directories. In addition, you may link both files and directories across different file systems. A symbolic link is actually a distinct file that contains a pointer to another file or directory. This pointer is the pathname to the destination file or directory. Only the original filename is the real name of the file or directory. Unlike a hard link, a soft link is actually only a link.

With both hard and soft links, changes made to a file through one name appear in the file as seen through another name.

A major difference between hard and soft links occurs when removing them. A file with hard-linked names persists until all its names have been removed. A file with soft-linked names vanishes when its original name has been removed; any remaining soft links then point to a nonexistent file. See Section 3.4.5.

3.4.2 Links and File Systems

The term **file system** as used in this discussion of links differs from its earlier usage in this book. Previously, a file system was defined as a useful arrangement of files into a directory structure. Here, the same term acquires a more precise meaning: the files and directories contained within a single **disk partition**. A disk partition is a physical disk, or a portion of one, that has been prepared to contain file directories.

You can use the `df` command to discover the name of the disk partition that holds any particular directory on your operating system. Here is an example in which `df` shows that the directories `/u1/info` and `/etc` are in different file systems, but that `/etc` and `/tmp` are in the same file system:

```

$ df /u1/info
Filesystem 512-blks      used  avail capacity  Mounted on
/dev/rz2c   196990 163124 14166    92%    /u1
$ df /etc
Filesystem 512-blks      used  avail capacity  Mounted on
/dev/rz3a   30686  19252  8364    70%    /
$ df /tmp
Filesystem 512-blks      used  avail capacity  Mounted on
/dev/rz3a   30686  19252  8364    70%    /
$

```

3.4.3 Using Links

To link files in the same file system, use the following command format:

```
ln /dirname1/filename1 /dirname2/filename2
```

The */dirname1/filename1* entry is the pathname of an existing file. The */dirname2/filename2* entry is the pathname of a new filename to be linked to the existing */dirname1/filename1*. The *dirname1* and *dirname2* arguments are optional if you are linking files in the same directory.

If you want to link files and directories across file systems, you can create **symbolic links**. To create a symbolic link, add an `-s` flag to the `ln` command sequence and specify the full pathnames of both files. The `ln` command for symbolic links takes the following form:

```
ln -s /dirname1/filename1 /dirname2/filename2
```

The */dirname1/filename1* entry is the pathname of an existing file. The */dirname2/filename2* entry is a pathname of a new filename in a different file system.

In the following example, the `ln` command links the new filename `checkfile` to the existing file named `file3`:

```

$ ln file3 checkfile
$

```

Now use the `more` command to verify that `file3` and `checkfile` are two names for the same file:

```
$ more file3
```

The system displays the following:

```
You will find that vi is a useful
editor that has many features.
$
```

Now display the text of checkfile:

```
$ more checkfile
You will find that vi is a useful
editor that has many features.
$
```

Notice that both `file3` and `checkfile` contain the same information. Any change that you make to the file under one name will show up when you access the file by its other name. Updating `file3`, for example, will also update `checkfile`.

If your two files were located in directories that are in two different file systems, you need to create a symbolic link between them. For example, to link a file called `newfile` that is in the `/reports` directory to the file called `mtgfile` in the `/summary` directory, you can create a symbolic link by using the following:

```
$ ln -s /reports/newfile /summary/mtgfile
$
```

The information in both files is still updated in the same manner as previously explained.

3.4.4 How Links Work – Understanding Filenames and i-numbers

Each file has a unique identification number, called an **i-number**. The i-number refers to the file itself – data stored at a particular location – rather than to the filename. The i-number distinguishes the file from other files within the same file system.

A directory entry is a link between an i-number that represents a physical file and a filename. It is this relationship between files and filenames that enables you to link multiple filenames to the same physical file – that is, to the same i-number.

To display the i-numbers of files in your current directory, use the `ls` command with the `-i` (print i-number) flag in the following format:

```
ls -i
```

Examine the identification numbers of the files in your login directory. The number preceding each filename in the listing is the i-number for that file.

```
$ ls -i
1079 checkfile 1077 file1 1078 file2 1079 file3
$
```

The i-numbers in your listing will differ from those shown in this example. However, the important thing to note is the identical i-numbers for `file3` and `checkfile`, the two files linked in the previous example. In this case, the i-number is 1079.

Because an i-number represents a file within a particular filesystem, hard links cannot exist between separate file systems.

The situation is entirely different with symbolic links, where the link becomes a new file with its own, new i-number. The symbolic link is not another filename on the original file's i-number, but instead is a separate file with its own i-number. Because the symbolic link refers to the original file by name, rather than by i-number, symbolic links work correctly between separate file systems.

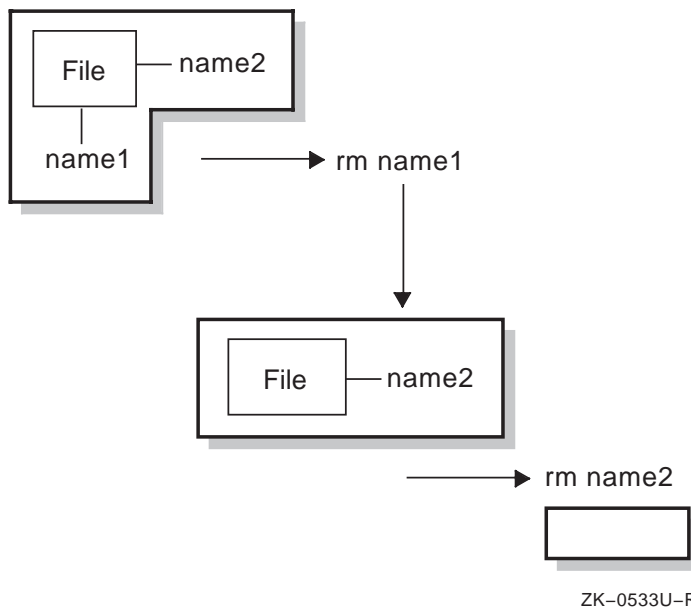
3.4.5 Removing Links

The `rm` (remove file) command does not always remove a file. For example, suppose that a file is linked to more than one filename; that is, several names refer to the same i-number. In this case, the `rm` command removes the link between the i-number and that filename, but leaves the physical file intact.

The `rm` command actually removes a physical file only after it has removed the last link between that file and a filename, as shown in Figure 3-1. When a symbolic link is removed, the filename specifying the pointer to the destination file or directory is removed.

For detailed information about the `rm` command, refer to Section 3.9.

Figure 3-1: Removing Links and Files



To display both the i-numbers and the number of filenames linked to a particular i-number, use the `ls` command with the `-i` (print i-number) and the `-l` (long listing) flags in the following format:

ls -il

Examine the links in your login directory. Remember that the i-numbers displayed on your screen will differ from those shown in the example and that your username and your group's name will replace the `larry` and `system` entries.

```
$ ls -il
total 3
1079 -rw-r--r-- 2 larry system 65 Jun 5 10:06 checkfile
1077 -rw-r--r-- 1 larry system 101 Jun 5 10:03 file1
1078 -rw-r--r-- 1 larry system 75 Jun 5 10:03 file2
1079 -rw-r--r-- 2 larry system 65 Jun 5 10:06 file3
1080 drwxr-xr-x 2 larry system 32 Jun 5 10:07 project
$
```

Again, the first number in each entry shows the i-number for that filename. The second element in each line shows the file permissions, described in detail in Chapter 5.

The third field for each entry, the number to the left of the username, represents the number of links to that i-number. Notice that `file3` and `checkfile` have the same i-number, 1079, and that both show two links. Each time the `rm` command removes a filename, it reduces the number of links to that i-number by one.

In the following example, use the `rm` command to remove the filename `checkfile`.

```
$ rm checkfile
$
```

List the contents of the directory with the `ls -il` command. Notice that the `rm` command has reduced the number of links to i-number 1079, which is the same i-number to which `file3` is linked, by one.

```
$ ls -il
total 3
1077 -rw-r--r-- 1 larry system 101 Jun 5 10:03 file1
1078 -rw-r--r-- 1 larry system  75 Jun 5 10:03 file2
1079 -rw-r--r-- 1 larry system  65 Jun 5 10:06 file3
1080 drwxr-xr-x 2 larry system  32 Jun 5 10:07 project
$
```

3.5 Copying Files (cp)

This section provides information about how to copy files on a local system. For information about copying files to and from remote systems see Chapter 12 and Chapter 14.

The `cp` (copy) command copies a file from one filename to another filename in your current directory or copies the file from one directory to another directory.

The `cp` command is especially useful to make backup copies of important files. Because the backup and the original are two distinct files, you can make changes to the original while still maintaining an unchanged copy in the backup file. This is helpful in case something happens to the original version. Also, if you decide you do not want to save your most recent changes to the original file, you can begin again with the backup file.

Compare the `cp` command, which actually copies files, with the `ln` command, which creates multiple names for the same file. Section 3.4 explains the `ln` command in detail. Refer also to the `cp(1)` and `ln(1)` reference pages.

The format of the `cp` command is:

```
cp source destination
```

The *source* entry is the name of the file to be copied. The *destination* entry is the name of the file to which you want to copy *source*. The *source* and *destination* entries can be filenames in your current directory or pathnames to different directories. This statement is true when you are copying files from one directory to another. To copy the contents of an entire directory to another directory (recursively, using the `-r` option), see Section 4.4.

To copy files to a different directory, use the general format of the `cp` command. In this case, *source* is a series of one or more filenames and *destination* is a pathname that ends with the name of the target directory. In the *source* entry you may also use pattern-matching characters.

3.5.1 Copying Files in the Current Directory

The `cp` command creates the destination file if it does not already exist. However, if a file with the same name as the destination file does exist, `cp` copies the source file over the existing destination file.

Caution

If the destination file exists, your shell may allow the `cp` command to erase the contents of that file before it copies the source file. As a result, be certain that you do not need the contents of the destination file, or that you have a backup copy of the file, before you use it as the destination file for the `cp` command. If you use the C shell, see Table 8-6 for the *noclobber* variable that can be set to prevent the erasure of the destination file.

In the following example, the destination file does not exist, so the `cp` command creates it. First, list the contents of your login directory:

```
$ ls
file1  file2  file3  project
$
```

Copy the source file, `file2`, into the new destination file, `file2x`:

```
$ cp file2 file2x
$
```


List the contents of the directory to verify that the copying process was successful:

```
$ ls
file1  file2  file2x  file3  project
$
```

3.5.2 Copying Files into Other Directories

You need a subdirectory to work through the following example, so create one called `reports` with the `mkdir` command:

```
$ mkdir reports
$
```

To copy the `file2` file into the `reports` directory, enter:

```
$ cp file2 reports
$
```

List the contents of `reports` to verify that it contains a copy of `file2`:

```
$ ls reports
file2
$
```

You can also use the `cp` command to copy multiple files from one directory into another directory. The format of the command is:

```
cp filename1 filename2 dirname
```

In the following example, enter the `cp` command to copy both `file2` and `file3` into the `reports` directory, and then list the contents of that directory:

```
$ cp file2 file3 reports
$ ls reports
file2 file3
$
```

In the above example, you do not have to specify `file2` and `file3` as part of the `dirname` entry because the files being copied are retaining their original filenames.

You may also use pattern-matching characters to copy files. For example, to copy `file2` and `file3` into `reports`, enter:

```
$ cp file* reports
$
```

To change the name of a file when you copy it into another directory, enter the name of the source file (the original file), the directory name, a slash (/), and then the new filename. In the following example, copy `file3` into the `reports` directory under the new name `notes`, and list the contents of the `reports` directory:

```
$ cp file3 reports/notes
$ ls reports
file2 file3 notes
$
```

3.6 Renaming or Moving Files (mv)

You can use the `mv` (move) command to perform the following actions:

- Move one or more files from one directory into another directory
- Rename files

The format of the `mv` command is:

```
mv oldfilename newfilename
```

The *oldfilename* entry is the name of the file you want to move or rename. The *newfilename* entry is the new name you want to assign to the original file. Both entries can be names of files in the current directory, or pathnames to files in a different directory. You may also use pattern-matching characters.

The `mv` command links a new name to an existing i-number and breaks the link between the old name and that i-number. It is useful to compare the `mv` command with the `ln` and `cp` commands, which are explained in Section 3.4 and Section 3.5. Refer also to the reference pages for these commands.

3.6.1 Renaming Files

In the following example, first list the i-number of each file in your current directory with the `ls -i` command. Next, enter the `mv` command to change the name of file `file2x` to `newfile`. The i-numbers displayed on your screen will differ from the numbers in the example:

```
$ ls -i
1077 file1      1088 file2x    1080 project
1078 file2      1079 file3    1085 reports
$ mv file2x newfile
$
```

Again, list the contents of the directory:

```
$ ls -i
1077 file1    1079 file3    1080 project
1078 file2    1088 newfile   1085 reports
$
```

Note two things in this example:

- The `mv` command changes the name of file `file2x` to `newfile`.
- The `i`-number for the original file (`file2x`) and `newfile` is the same – 1088.

The `mv` command removes the connection between `i`-number 1088 and filename `file2x`, replacing it with a connection between `i`-number 1088 and filename `newfile`. However, the command does not change the file itself.

3.6.2 Moving Files into a Different Directory

You can also use the `mv` command to move one or more files from your current directory into a different directory.

Note

Type the target directory name carefully because the `mv` command does not distinguish between filenames and directory names. If you enter an invalid directory name, the `mv` command takes that name as a new filename. The result is that the file is renamed rather than moved.

In the following example, the `ls` command lists the contents of your login directory. The `mv` command moves `file2` from your current directory into the `reports` directory. The `ls` command then verifies that the file has been removed:

```
$ ls
file1  file2  file3  newfile  project  reports
$ mv file2 reports
$ ls
file1  file3  newfile  project  reports
$
```

List the contents of the `reports` directory to verify that the command has moved the file:

```
$ ls reports
file2  file3  notes
$
```

You may also use pattern-matching characters to move files. For example, to move `file1` and `file3` into `reports`, you could enter the following command:

```
$ mv file* reports
$
```

Now list the contents of your login directory to verify that `file1` and `file3` have been moved:

```
$ ls
newfile project reports
$
```

Copy `file1`, `file2`, and `file3` back into your login directory. The dot (`.`) in the following command line specifies the current directory, which in this case is your login directory:

```
$ cp reports/file* .
$
```

Verify that the files are back in your login directory:

```
$ ls
file1 file2 file3 newfile project reports
$
```

Lastly, verify that `file1`, `file2`, and `file3` are still in the `reports` directory:

```
$ ls reports
file1 file2 file3 newfile project reports
$
```

3.7 Comparing Files (diff)

You can compare the contents of text files with the `diff` command. Use the `diff` command when you want to pinpoint the differences in the contents of two files that are expected to be somewhat different.

The format of the `diff` command is:

```
diff file1 file2
```

The `diff` command scans each line in both files looking for differences. When the `diff` command finds a line (or lines) that differ, for each line that is different the following information is reported:

- Line numbers of any changes
- Whether the difference is an addition, a deletion, or a change to the line

If the change is caused by an addition, `diff` displays:

```
l[,l] a r[,r]
```

where `l` is a line number in `file1` and `r` is a line number in `file2`. The `a` indicates an addition. If the difference was a deletion, `diff` would specify a `d`; if the difference was a change to a line, `diff` would specify a `c`.

The actual differing lines follow. In the leftmost column, a left angle bracket (<) indicates lines from `file1`, and a right angle bracket (>) indicates lines from `file2`.

For example, suppose that you want to quickly compare the following meeting rosters in the files `jan15mtg` and `jan22mtg`:

<code>jan15mtg</code>	<code>jan22mtg</code>
alice	alice
colleen	brent
daniel	carol
david	colleen
emily	daniel
frank	david
grace	emily
helmut	frank
howard	grace
jack	helmut
jane	jack
juan	jane
lawrence	juan
rusty	lawrence
soshanna	rusty
sue	soshanna
tom	sue
	tom

Instead of tediously comparing the list by sight, you can use the `diff` command to compare `jan15mtg` with `jan22mtg` as follows:

```
$ diff jan15mtg jan22mtg
2a3,4
> brent
> carol
10d11
< howard
$
```

Here we find that Brent and Carol attended the meeting on January 22, and Howard did not. We know this because the line number and text output indicate that `brent` and `carol` are additions to file `jan22mtg` and that `howard` is a deletion.

In cases where there are no differences between files, the system will merely return your prompt. For more information, see the `diff(1)` reference page.

3.8 Sorting File Contents (`sort`)

You can sort the contents of text files with the `sort` command. You can use this command to sort a single file or multiple files.

The format of the `sort` command is:

```
sort filename
```

The *filename* entry can be the name of the file, the relative pathname of the file, the full pathname of the file, or a list of filenames separated by spaces. You may also use pattern-matching characters to specify files. See Chapter 2 for information about pattern matching.

A good example of what the `sort` command can do for you is to sort a list of names and put them in collated order as defined by your current locale. For example, assume that you have lists of names that are contained in three files, `list1`, `list2`, and `list3`:

<code>list1</code>	<code>list2</code>	<code>list3</code>
Zenith, Andre	Rocca, Carol	Hamilton, Abe
Dikson, Barry	Shepard, Louis	Anastio, William
D'Ambrose, Jeanette	Hillary, Mimi	Saluccio, William
Julio, Annette	Chung, Jean	Hsaio, Peter

To sort the names in all three files, enter:

```
$ sort list*
Anastio, William
Chung, Jean
D'Ambrose, Jeanette
Dickson, Barry
Hamilton, Abe
Hillary, Mimi
Hsaio, Peter
Julio, Annette
Rocca, Carol
Saluccio, Julius
Shepard, Louis
Zenith, Andrew
$
```

You can also capture the sorted list by redirecting the screen output to a file that you name by entering:

```
$ sort list* >newlist
$
```

For more information about redirecting output, see Chapter 6. For a detailed description of the `sort` command and its many options, see the `sort(1)` reference page.

3.9 Removing Files (rm)

When you no longer need a file, you can remove it with the `rm` (remove file) command. Use this command to remove a single file or multiple files.

The format of the `rm` command is:

```
rm filename
```

The *filename* entry can be the name of the file, the relative pathname of the file, the full pathname of the file, or a list of filenames. The format you use depends on where the file is located in relation to your current directory.

3.9.1 Removing a Single File

In the following example, you remove the file called `file1` from your login directory.

First, return to your login directory with the `cd` (change directory) command. Next, enter the `pwd` (print working directory) command to verify that your login directory is your current directory, and then list its contents. Remember that the system substitutes the name of your login directory for the notation `/u/uname` in the example.

```
$ cd
$ pwd
/u/uname
$ ls
file1  file2  file3  newfile  project  reports
$
```

Enter the `rm` command to remove `newfile`, and then list the contents of the directory to verify that the system has removed the file.

```
$ rm newfile
$ ls
file1  file2  file3  project  reports
$
```

You must have permission to access a directory before you can remove files from it. For information about directory permissions, see Chapter 5.

Note

In addition to removing one or more files, `rm` also removes the links between files and filenames. The `rm` command actually removes the file itself only when it removes the last link to that file. For information about using the `rm` command to remove links, see Section 3.4.5.

3.9.2 Removing Multiple Files – Matching Patterns

You can remove more than one file at a time with the `rm` command by using pattern-matching characters. See Chapter 2 for a description of pattern-matching characters.

For example, suppose your current directory contains the following files: `receivable.jun`, `payable.jun`, `payroll.jun`, and `expenses.jun`. You can remove all four of these files with the `rm *.jun` command.

Caution

Be certain that you understand how the `*` pattern-matching character works before you use it. For example, the `rm *` command *removes every file* in your current directory. Be especially careful with `*` at the beginning or end of a filename. If you mistakenly enter `rm * name` instead of `rm *name`, you will remove all your files, rather than just those ending with *name*. (If your system is backed up on a regular basis, your system administrator can help you recover lost files.)

You may prefer to use the `-i` flag with the `rm` command, which prompts you for verification before deleting a file or files. See the end of this section for details.

You can also use the pattern-matching question mark (`?`) character with the `rm` command to remove files whose names are the same, except for a single character. For example, if your current directory contains the files `record1`, `record2`, `record3`, and `record4`, you can remove all four files with the `rm record?` command.

For detailed information about pattern-matching characters, see Chapter 2.

When using pattern-matching characters, you may find the `-i` (interactive) flag of the `rm` command particularly useful. The `rm -i` command allows you to selectively delete files. For each file selected by the command, the operating system asks whether or not you want to delete or retain the file.

For example, suppose that your directory contains the files `record1`, `record2`, `record3`, `record4`, `record5`, and `record6`. Create those files now in your login directory by using the `touch` command as follows:

```
$ touch record1 record2 record3 record4 record5 record6
$
```

The `touch` command is useful when you want to create empty files, as you are now. For complete information on the `touch` command, see the `touch(1)` reference page.

If you want to remove four of the six files that begin with the characters `record`, enter:

```
$ rm -i record?
rm: remove record1?n
rm: remove record2?y
rm: remove record3?y
rm: remove record4?y
rm: remove record5?y
rm: remove record6?n
$
```

In the preceding example, you have deleted all files except for `record1` and `record6`.

Note

In addition to removing one or more files, the `rm` command also provides an option, the `-r` flag, that removes files and directories at the same time. See Chapter 4 for more information.

3.10 Determining File Type (`file`)

Use the `file` command when you want to see what kind of data a file contains without having to display its contents. The `file` command displays whether the file is one of the following:

- A text file
- A directory
- Input for one of the text formatting packages `troff`, `nroff`, or `eqn` input text
- Source code for the C or FORTRAN programming languages
- An executable file

The `file` command is especially useful when you suspect that a file contains a compiled program. Displaying the contents of a compiled program can produce disconcerting results on your screen.

The format of the `file` command is:

file *filename*

The *filename* entry can be the name of the file, the relative pathname of the file, the full pathname of the file, or a list of filenames. The format you use depends on where the file is located in relation to your current directory. You may also use pattern-matching characters to specify files. See Chapter 2 for information on pattern matching.

For example, to determine the file type of entries in your login directory, enter the following:

```
$ cd
$ pwd
/u/uname
$ file *
file1:  ascii text
file2:  ascii text
file3:  ascii text
project: directory
record1: empty
record6: empty
reports: directory
$
```

The `file` command has identified `file1`, `file2`, and `file3` as ASCII text files, `project` and `reports` as directories, and `record1` and `record6` as empty files.

For more information on the `file` command, see the `file(1)` reference page.

Managing Directories 4

This chapter shows you how to manage directories on your system. After completing this chapter, you will be able to:

- Create directories
- Change directories
- Display, copy, and rename directories
- Remove directories

To learn about managing directories, try the examples in this chapter. You should perform each example in sequence so that the information on your screen is consistent with the information shown in this chapter.

Before you can do the examples, you must be logged in and your login directory should be in the state that you left it after doing the examples in Chapter 3. As a result, your login directory should contain the following:

- The files `file1`, `file2`, `file3`, `record1`, and `record6`
- The subdirectory, `reports`, that contains the files `file1`, `file2`, `file3`, and `notes`
- The empty subdirectory `project`

If you are using files with different names, make the appropriate substitutions as you work through the examples. To produce a listing of the files in your current directory, use the `ls` command, which is explained in Chapter 3.

4.1 Creating a Directory (`mkdir`)

Directories allow you to organize individual files into useful groups. For example, you could put all the sections of a report in a directory named `reports`, or the data and programs you use in cost estimating in a directory named `estimate`. A directory can contain files, other directories, or both.

Your login directory was created for you when your computer account was established. However, you will probably need additional directories to organize the files you create while working with the operating system. You create new directories with the `mkdir` (make directory) command.

The format of the `mkdir` command is:

```
mkdir dirname
```

The *dirname* entry is the name you want to assign to the new directory.

The system creates *dirname* as a subdirectory of your working directory. This means that the new directory is located at the next level below your current directory.

In the following example, return to your login directory by entering the `cd` command, and create a directory named `project2`:

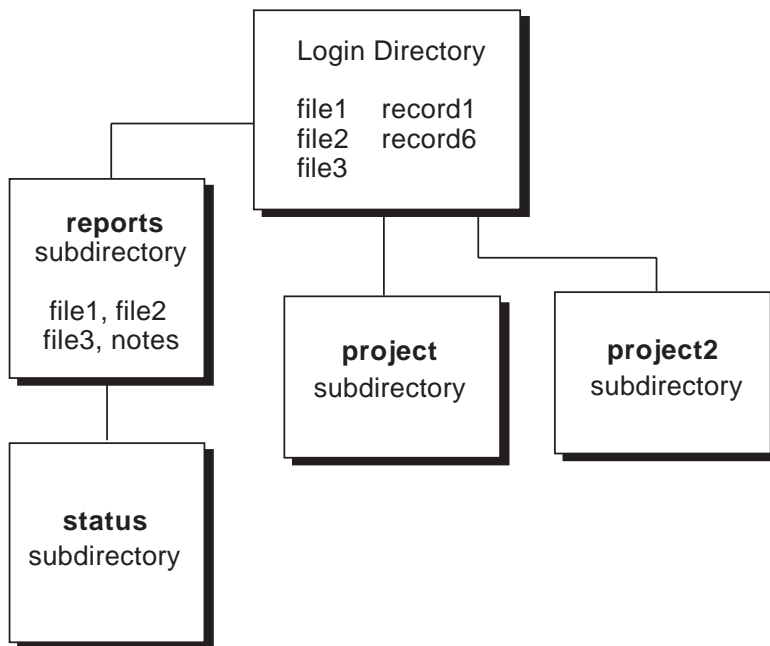
```
$ cd  
$ mkdir project2  
$
```

Now, create a subdirectory in the `reports` directory by entering a relative pathname:

```
$mkdir reports/status  
$
```

Figure 4-1 shows the new file system tree structure. The `project`, `project2`, and `reports` directories are located one level below your login directory, and the `status` subdirectory is located one level below the `reports` directory.

Figure 4-1: Relationship Between Directories and Subdirectories



ZK-0534U-R

Like filenames, the maximum length of a directory name depends upon the file system used on your computer. For example, your file system may allow a maximum directory name length of 255 bytes (the default), or it may allow a maximum directory name length of only 14 bytes. Knowing the maximum directory name length is important to help you give meaningful names to your directories. See your system administrator for details.

The operating system does not have a symbol or notation that automatically distinguishes between a filename and a directory name, so you may find it useful to establish your own naming conventions to designate files and directories. However, you can use the `ls -F` command to distinguish between filenames and directory names when the contents of your current directory are displayed. For more information on this command, see Section 4.3.

4.2 Changing Directories (cd)

The `cd` (change directory) command allows you to switch from your current (working) directory to another directory. You can move to any directory in the file system from any other directory in the file system by executing `cd` with the proper pathname.

Note

You must have execute permission to access a directory before you can use the `cd` command. For information about directory permissions, see Chapter 5.

The format of the `cd` command is:

cd *pathname*

The *pathname* entry can either be the full pathname or the relative pathname of the directory that you want to set as your current directory.

If you enter the `cd` command without a pathname, the system returns you to your login directory (also known as your HOME directory).

To check the name of and display the path for your current directory, enter the `pwd` (print working directory) command. See Chapter 2 for information about the `pwd` command.

4.2.1 Changing Your Current Directory

In the following example, you enter the `pwd` command to display the name (which is also the pathname) of your working directory. You then use the `cd` command to change your current directory.

First return to your login directory, if necessary, by entering the `cd` command without a pathname. Next, enter the `pwd` command to verify that your login directory is your current directory. Remember that the system substitutes the name of your login directory for the notation `/u/uname` in the example:

```
$ cd
$ pwd
/u/uname
$
```


Enter the `cd` command with the relative pathname `project2` to change to the `project2` directory:

```
$ cd project2
$
```

Enter `pwd` again to verify that `project2` is the current directory. Then, enter `cd` to return to your login directory:

```
$ pwd
/u/uname/project2
$ cd
$
```

To change your current directory to the `status` directory, which is a different branch of the file system tree structure, enter the `cd` command with a full pathname:

```
$ cd reports/status
$ pwd
/u/uname/reports/status
$
```

4.2.2 Using Relative Pathname Notation

You can use the following relative pathname notation to change directories quickly:

- Dot notation (`.` and `..`)
- Tilde notation (`~`)

Every directory contains at least two entries that are represented by dot (`.`) and dot dot (`..`). These entries refer to directories relative to the current directory:

dot (<code>.</code>)	This entry refers to the current directory.
dot dot (<code>..</code>)	This entry refers to the parent directory of your working directory. The parent directory is the directory immediately above the current directory in the file system tree structure.

To display the `.` and `..` entries as well as any files beginning with a period, use the `-a` flag with the `ls` command.

In the following example, change to the `reports` directory by changing first to your login directory and then to the `reports` directory:

```
$ cd
$ cd reports
$
```

The `ls` command displays the directory contents as well as the `status` subdirectory you created earlier:

```
$ ls
file1  file2  file3  notes  status
$
```

Now, execute the `ls -a` command to list all directory entries as well as those that begin with a dot (`.`) – the relative directory names:

```
$ ls -a
./  ../  file1  file2  file3  notes  status
$
```

You can use the relative directory name dot dot (`..`) to refer to files and directories located above the current directory in the file system tree structure. That is, if you want to move up the directory tree one level, you can use the relative directory name for the parent directory rather than using the full pathname.

In the following example, the `cd ..` command changes the current directory from `reports` to your login directory, which is the parent directory of `reports`. Remember that the `/u/username` entry represents your login directory.

```
$ pwd
/u/username/reports
$ cd ..
$ pwd
/u/username
$
```

To move up the directory structure more than one level, you can use a series of relative directory names, as shown in the following example. The response to the following `pwd` command, the slash (`/`) entry, represents the root directory.

```
$ cd ../..
$ pwd
/
$
```

In the C shell and the Korn shell, you may use a tilde (~) to specify a user's login directory. For example, to specify your own login directory, use the tilde alone as follows:

```
$ cd ~
```

The above tilde notation does not save you keystrokes because in all operating system shells you may get the same results by merely entering `cd` from any place in the file system.

However, if you want to access a directory below your login directory, tilde notation can save you keystrokes. For example, to access the `reports` directory from anywhere in the file system, enter the following:

```
$ cd ~/reports
```

Tilde notation is also very useful when you want to access a file or directory either in or below another user's login directory. You may not know the precise location of that user's login directory, but assuming you have the appropriate permissions, you could get there with a minimum of keystrokes.

For example, from any place in the file system, you could specify the login directory of a hypothetical user `jones` by entering the following:

```
$ cd ~jones
```

In addition, if user `jones` tells you that you can find a file in the `status` directory immediately below the login directory, you can access the directory by entering the following:

```
$ cd ~jones/status
```

4.2.3 Accessing Directories Through Symbolic Links

When directories are connected through a symbolic link, the parent directory you access with the `cd` command differs depending upon whether you are specifying the actual directory name or the relative directory name. In particular, using the full pathname to find the parent of a symbolically linked directory results in accessing the actual parent directory.

For example, suppose `user2` is working on a file in the `/u/user2/project` directory, which is the symbolic link to `/u/user1/project`. To change to the actual parent directory (`/u/user2`), `user2` types the following:

```
$ cd /u/user2
$ pwd
/u/user2
$
```

If `user2` specified the relative directory name (`..`), the parent directory of the symbolic link would be accessed. For example, suppose `user2` is working on the same file in the `/u/user2/project` directory, which is the symbolic link to `/u/user1/project`. To access the parent directory of the symbolic link, `user2` enters the following:

```
$ cd ..
$ pwd
/u/user1
$
```

Instead of being in the `/u/user2` directory, `user2` is now in the directory called `/u/user1`.

For background information on symbolic links, see Section 3.4.

4.3 Displaying Directories (`ls -F`)

A directory can contain subdirectories as well as files. To display subdirectories, use the `ls -F` command. This command displays the contents of the current directory and marks each directory with a trailing slash character (`/`) so that it can be readily distinguished from a file.

The format of the `ls -F` command is:

ls -F

In the following example, return to your login directory and enter the `ls -F` command to display the directory contents. Note that the `project`, `project2`, and `reports` directories are marked with a slash:

```
$ cd
$ ls -F
file1      file3      project2/  record6
file2      project/   record1    reports/
$
```

Some C and Korn Shell users define an alias for the `ls` command so that whenever they enter `ls`, the `ls -F` command is executed. For more information about defining aliases, see Chapter 8.

4.4 Copying Directories (`cp`)

You can use the `cp` command with the `-r` flag to recursively copy directories and directory trees to another part of the file system. The `cp -r` command has the following format:

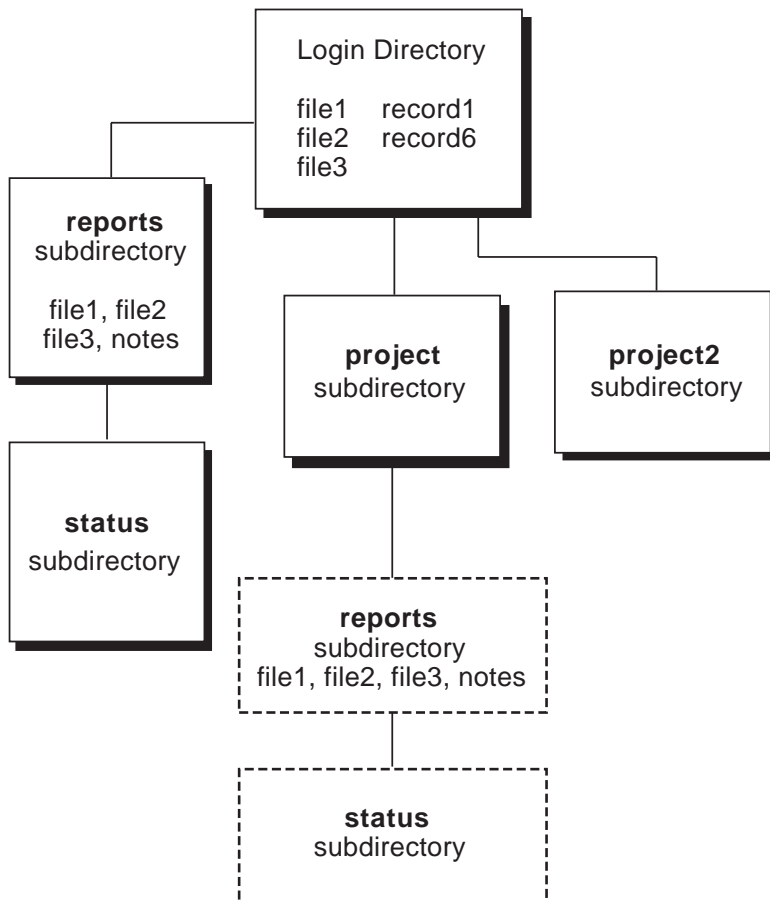
```
cp -r source destination
```

The *source* entry is the name of the directory to be copied. The *destination* entry is the name of the directory location to which you want to copy *source*.

Figure 4-2 shows how the `cp -r` command in the following example copies the directory tree `reports` into the directory `project`. It is assumed that the command is entered from the `login` directory:

```
$ cp -r reports project
$
```

Figure 4-2: Copying a Directory Tree



ZK-0535U-R

Note that the `reports` directory files, `file1`, `file2`, `file3`, and `notes`, as well as the `status` subdirectory, have been copied to `project`.

4.5 Renaming Directories (mv)

You can use the `mv` command to rename a directory *only* when that directory is contained in the same disk partition.

The format of the `mv` command is:

```
mv olddirectoryname newdirectoryname
```

The *olddirectoryname* entry is the name of the directory you want to move or rename. The *newdirectoryname* entry is the new name you want to assign to the original directory name.

In the following example, first change to the `reports` directory. Then, enter `ls -i -d` command to list the i-number for the `status` directory:

```
$ cd reports
$ ls -i -d status
1091 status
$
```

Now, enter the `mv` command to change the name of `status` to `newstatus`. Then, list the i-number for the `newstatus` directory:

```
$ mv status newstatus
$ ls -i -d newstatus
1091 newstatus
$
```

Notice that the second `ls -i -d` command does not list the original directory name `status`. However, it does list the new directory name, `newstatus`, and displays the same i-number (1091 in this example) for the new directory as for the original `status` directory.

4.6 Removing Directories (rmdir)

When you no longer need a particular directory, you can remove it from the file system with the `rmdir` (remove directory) command. This command removes only empty directories – those that contain no files or subdirectories. For information about removing files from directories, see Section 4.6.4 and Section 3.9.

The format of the `rmdir` command is:

```
rmdir dirname
```

The *dirname* entry is the name, or pathname, of the directory you want to remove.

Before working through the examples in the following sections, create three subdirectories in the directory `project2`.

First, use the `cd project2` command to set `project2` as your current directory. Next, use the `mkdir` command to create the `schedule`, `tasks`, and `costs` directories. Then, list the contents of the `project2` directory:

```
$ cd project2
$ mkdir costs schedule tasks
$ ls -F
costs/  schedule/  tasks/
$
```

Finally, use the `cd` command to return to your login directory:

```
$ cd
$ pwd
/u/uname
$
```

4.6.1 Removing Empty Directories

The `rmdir` command removes only empty directories. If you try to remove a directory that contains any files or subdirectories, the `rmdir` command displays an error message, as the following example shows:

```
$ rmdir project2
rmdir: project2 not empty
$
```

Note

You cannot remove a directory while you are positioned in it. To remove a directory, you must be elsewhere in the directory tree. See Section 4.6.3 for more information.

Before you can remove the directory `project2`, you must first remove the contents of that directory. In the following example, the `cd` command makes `project2` your current directory, and the `ls -F` command lists the contents of `project2`:

```
$ cd project2
$ ls -F
costs/  schedule/  tasks/
```

Now remove the directory `schedule` from the current directory, and then list the remaining contents of the `project2` directory:

```
$ rmdir schedule
$ ls -F
costs/  tasks/
$
```

The `project2` directory still contains two subdirectories: `costs` and `tasks`. You can remove them by using pattern-matching characters, as described in the next section. Once these subdirectories are removed, you can delete the `project2` directory, as described in Section 4.6.3.

4.6.2 Removing Multiple Directories

You can remove more than one directory at a time with the `rmdir` command by using pattern-matching characters. See Chapter 2 for detailed information about pattern-matching characters.

For example, suppose that you are in the `project2` directory and want to remove two subdirectories: `costs` and `tasks`. To do so, enter the `rmdir *s?s` command. Then, enter the `ls` command to verify that the `project2` directory contains no entries:

```
$ rmdir *s?s
$ ls
$
```

Caution

Entering the `rmdir` command with the asterisk (*) character alone removes ALL empty directories from your current directory. Use the asterisk (*) pattern-matching character with care.

4.6.3 Removing Your Current Directory

You cannot remove your current directory while you are still working in it. You can remove it only after you move into another directory. You generally enter the dot dot (`..`) command to move into the parent directory of your current directory, and then enter `rmdir` with the pathname of the target directory.

The directory `project2` is empty. To remove `project2`, first move to your login directory, which is the parent directory of `project2`. Then, use the `rmdir dirname` command to remove `project2`, and enter `ls` to confirm the removal:

```
$ cd
$ rmdir project2
$ ls
file1  file2  file3  project/  record1  record6  reports/
$
```


Your login directory no longer contains the `project2` directory.

4.6.4 Removing Files and Directories Simultaneously (`rm -r`)

The `rmdir` command removes only directories, not files. You can, however, remove files and directories at the same time by using the `rm` command with the `-r` (recursive) flag.

The `rm -r` command first deletes the files from a directory and then deletes the directory itself. It deletes the directory you specify as well as any subdirectories (and the files they contain) below it on the directory tree. This command should be used with caution.

The format of the `rm -r` command is:

```
rm -r pathname
```

The *pathname* entry can either be the full pathname or the relative pathname of the directory that you want to remove. You may also use pattern-matching characters to specify files.

Caution

Be certain that you understand how the `-r` flag works before you use it. For example, entering the `rm -r *` command from your login directory **deletes all files and directories to which you have access**. If you have superuser authority and are in the root directory, this command will *delete all system files*. See Section 5.7 for more information about superuser authority.

When using the `rm -r` command to remove files or directories, it is a good idea to include the `-i` flag in the command line:

```
rm -ri pathname
```

When you enter the command in this form, the system prompts you for verification before actually removing the specified item(s). In this way, by answering `y` (yes) or `n` (no) in response to the prompt, you control the actual removal of a file or directory. Keep in mind that using the `-ri` option may require you to reply to many, many prompts (depending upon how many files you have).

Controlling Access to Your Files and Directories **5**

This chapter shows you how to control access to your system as well as your files and directories. After reading this chapter, you will be able to:

- Understand password, group, and system security issues
- Understand file and directory permissions
- Display and set file and directory permissions
- Change owners and groups
- Change your identity to access files
- Understand superuser concepts
- Learn where to find information about enhancements to security that may be installed on your system

A good way to learn about the topics in this chapter is to do the examples so that the information on your screen is consistent with the information in this book.

Before you can work through the examples, you must be logged in and your login directory should be in the state that you left it after doing the examples in Chapter 4. Your login directory should contain:

- The files `file1`, `file2`, `file3`, `record1`, and `record6`
- The subdirectory `reports` that contains the `file1`, `file2`, `file3`, and `notes` files and the subdirectory `newstatus`
- The `project` subdirectory that contains the files `file1`, `file2`, `file3`, and `notes` as well as the subdirectory `status`

If you are using files with different names, make the appropriate substitutions as you work through the examples.

5.1 Understanding Password and Group Security Files

Before a user can log in successfully, the user must be made known to the system by the creation of a user account. Adding a user account is a routine but critical activity that is usually performed by the system administrator.

When a user account is created, information about the new user is added to the following two files:

`/etc/passwd` This file contains individual user information for all users of the system.

`/etc/group` This file contains group information for all groups on the system.

These files define who can use the system and each user's access rights. In addition, all other system security controls depend upon password and group security. The following sections describe the `/etc/passwd` and `/etc/group` files.

5.1.1 The `/etc/passwd` File

The `/etc/passwd` file contains records that define login accounts and attributes for all system users. This file can be altered only by a user with **superuser** privileges. See Section 5.7 for more information.

Each record in the `/etc/passwd` file defines a login account for an individual user. The fields are separated by colons and the last field ends with a newline character. The following text shows the format of an `/etc/passwd` file entry and describes the meaning of each field:

```
username:password:UID:GID:user_info:login_directory:login_shell
```

username Your login name.

password Your password stored in encrypted form. Encryption prevents unauthorized users or programs from discovering your actual password. If no password has been specified for a user, this field will be blank.

UID (User ID) A unique number identifying you to the system.

GID (Group ID) A number identifying your default group. You can belong to one or more groups.

user_info This field can contain the following: your full name, maximum file size (a number limiting the maximum size of any file you create or extend), and site specific information (an attribute serving various purposes for each installation - it normally records biographical information).

login_directory Your current directory after logging in to the system. It is usually a directory you own and use to store private files.

login_shell The program run by the `login` program after you successfully log in to the system. It is normally a shell

program used to interpret commands. For more information on shells, see Chapter 7 and Chapter 8.

The following example is a sample entry in the `/etc/passwd` file:

```
lee:NebPsa9qxMkbD:201:20:Lee Voy,sales,x1234:/users/lee: \  
/usr/bin/sh
```

The user account `lee` has user ID 201 and group ID 20. Lee's full name is Lee Voy, and his department and telephone are listed. The login directory is `/users/lee` and the Bourne shell (`/usr/bin/sh`) is defined as the command interpreter. The password field contains Lee's password in encrypted form.

5.1.2 The `/etc/group` File

The `/etc/group` file defines login accounts for all groups using the system. This file can be altered only by a user with superuser privileges. See Section 5.7 for more information.

Each record in the group database defines the login account of one group. Groups provide a convenient way to share files among users with a common interest or who are working on the same project.

Each entry in the `/etc/group` file is a single line that contains four fields. The fields are separated by colons, and the last field ends with a newline character. The following text shows the format of each entry and describes the meaning of each field:

```
groupname:password:GID:user1[, user2, ..., userN ]
```

groupname A unique character string that identifies the group to the system.

password This field is always empty. Entries in this field are ignored.

GID (Group ID) A unique number that identifies the group to the system.

usernames A list of users who belong to the group.

5.2 Protecting Files and Directories

The operating system has a number of commands that enable you to control access to your files and directories. You can protect a file or directory by setting or changing its **permissions**, which are codes that determine the way in which anyone working on your system can use the stored data.

Setting or changing permissions is also referred to as setting or changing the protections on your files or directories. You generally protect your data for one or both of the following reasons:

- Your files and directories contain sensitive information that should not be available to everyone who uses your system.
- Not everyone who has access to your files and directories should have the permission to alter them.

Caution

Your system may allow two or more users to make changes to the same file at the same time without informing them. If this is so, the system saves the changes made by the last user to close the file; changes made by the other users are lost (some text editors warn users of this situation). It is therefore a good idea to set file permissions to allow only authorized users to modify files. The specified users should then communicate about when and how they are using the files.

Each file and each directory has nine permissions associated with it. Files and directories have the following three types of permissions:

- `r` (read)
- `w` (write)
- `x` (execute)

These three permissions occur for each of the following three classes of users:

- `u` (user/owner)
- `g` (group)
- `o` (all others; also known as "world")

The `r` permission allows users to view or print the file. The `w` permission allows users to write to (modify) the file. The `x` permission allows users to execute (run) the file or to search directories.

The user/owner of a file or directory is generally the person who created it. If you are the owner of a file, you can change the file permissions with the `chmod` command, which is described in Section 5.4.

The `group` specifies the group to which the file belongs. If you are the owner of a file, you can change the group ID of the file with the `chgrp` command, which is described in Section 5.8.

Note

If you do not own a file, you cannot change its permissions or group ID unless you have superuser authority. See Section 5.7 for more information.

The meanings of the three types of permissions differ slightly between ordinary files and directories. See Table 5-1 for more information.

Table 5-1: Differences Between File and Directory Permissions

Permission	For a File	For a Directory
r (read)	Contents can be viewed or printed.	Contents can be read, but not searched. Normally r and x are used together.
w (write)	Contents can be changed or deleted.	Entries can be added or removed.
x (execute)	File can be used as a program.	Directory can be searched.

5.3 Displaying File and Directory Permissions (ls)

To display the current file permissions, enter the `ls` command with the `-l` flag. To display the permissions for a single file or selected files, enter the following command:

```
$ ls -l filename
```

The *filename* entry can be the name of the file or a list of filenames separated by spaces. You may also use pattern-matching characters to specify files. See Section 5.4.1.3 for more information.

To display the permissions for all of the files in your current directory, enter the `ls -l` command:

```
$ ls -l
total 7

-rw-r--r-- 1 larry system 101 Jun 5 10:03 file1
-rw-r--r-- 1 larry system 171 Jun 5 10:03 file2
-rw-r--r-- 1 larry system 130 Jun 5 10:06 file3
drwxr-xr-x 2 larry system 32 Jun 5 10:07 project
-rw-r--r-- 1 larry system 0 Jun 5 11:03 record1
-rw-r--r-- 1 larry system 0 Jun 5 11:03 record6
drwxr-xr-x 2 larry system 32 Jun 5 10:31 reports
$
```

The first string of each entry in the directory shows the permissions for that file or directory. For example, the fourth entry, `drwxr-xr-x`, shows the following:

- That this is a directory (the `d` notation)
- That the owner can view it, write in it, and search it (the `rw` sequence)
- That the group can view it and search it, but not write in it (the first `r-x` sequence)
- That all others can view it and search it, but not write in it (the second `r-x` sequence)

The third field shows the file's owner, (in this case, `larry`), and the fourth field shows the group to which the file belongs, (in this case, `system`).

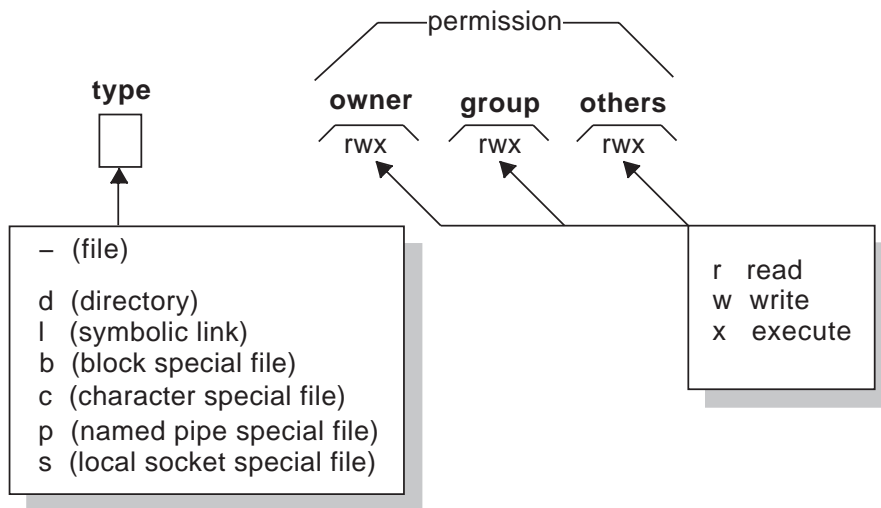
To list the permissions for a single directory, use the `ls -ld` command:

```
$ ls -ld reports
drwxr-xr-x  2 larry  system    32 Jun  5 10:31 reports
$
```

Taken together, all the permissions for a file or directory are called its **permission code**. As Figure 5-1 shows, a permission code consists of four parts:

- A single character shows the file type. The dash (`-`) indicates an ordinary file, `d` a directory, and `l` a symbolic link. Any other character indicates an I/O (Input/Output) device.
- A 3-character **permission field** shows user (owner) permissions, which may be any combination of read, write, and execute.
- Another 3-character permission field shows group permissions.
- Another 3-character permission field shows permissions for all others.

Figure 5-1: File and Directory Permission Fields



ZK-0536U-R

When you create a file or directory, the system automatically supplies a predetermined permission code. The following is a typical file permission code:

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

This file permission code specifies that the owner has read and write permissions while the group and all others have read permission. The dashes (-) in some positions following the file-type notation indicate that the specified class of user does not have permission for that operation.

The following is a typical directory permission code:

```
drwxr-xr-x
```

This directory permission code specifies that owner has read, write, and search permissions, while the group and all others have read and search permissions.

The default permission codes that your system provides relieve you from the task of specifying them explicitly every time you create a file or directory. If you want to create your own default permission codes, you must change your user mask with the `umask` command. For an explanation of the `umask` command, see the description of the command in Section 5.5.

5.4 Setting File and Directory Permissions (chmod)

Your ability to change permissions gives you a great deal of control over the way your data can be used. Use the `chmod` (change mode) command to set or change the permissions for your files and directories.

For example, you obviously permit yourself to read, modify, and execute a file. You generally permit members of your group to read a file. Depending upon the nature of your work and the composition of your group, you often allow them to modify or execute it. You generally prohibit all other system users from having any access to a file.

Note

You must be the owner of the file or directory (or have superuser authority) before you can change its permissions. This means that your username must be in the third field in an `ls -l` listing of that file.

It is important to realize that whatever restrictions you impose on file and directory access, the superuser can always override them. For example, if you use the `chmod` command to specify that only you can have access to the `report20` file. The superuser can still access this file. For more information on this topic, see Section 5.7.

There are two ways to specify the permissions set by the `chmod` command:

- You can specify permissions with letters and operation symbols.
- You can specify permissions with octal numbers.

The following sections describe how to specify permissions with letters and operation symbols, as well as with octal numbers.

5.4.1 Specifying Permissions with Letters and Operation Symbols

You can use letters and operation symbols to change file and directory permissions.

The following is the format of the `chmod` command when using letters and operation symbols:

chmod *userclass-operation-permission filename*

The *userclass-operation-permission* entry actually represents three codes that specify the user class, group, operation, and permission code that you want to activate. The *filename* entry is the name of the file or files whose permissions you want to change. You may also use pattern-matching characters to specify files. See Section 5.4.1.3 for more information.

User classes, operations, and permissions are defined as follows:

- Use one or more of these letters to represent the *userclass*:
 - u User (owner)
 - g Group
 - o All others (besides owner and group)
 - a All (user, group, and all others)
- Use one of these symbols to represent the *operation*:
 - + Add permission
 - Remove permission
 - = Assign permission regardless of previous setting
- Use one or more of these letters to represent the type of *permission*:
 - r Read
 - s Set user or group ID
 - w Write
 - x Execute

5.4.1.1 Changing File Permissions

In the following example, first enter the `ls -l` command to display the permissions for the `file1` file:

```
$ ls -l file1
-rw-r--r-- 1 larry system 101 Jun 5 10:03 file1
$
```

The owner (larry) has read/write permissions while the group and others have only read permissions.

Now, enter the `chmod` command with the flags `go+w`. This command expands the permissions for both the group (`g`) and for others (`o`) by giving them write access (`+w`) to `file1` in addition to the read access they already have:

```
$ chmod go+w file1
$
```

Next, list the new permissions for the file:

```
$ ls -l file1
-rw-rw-rw- 1 larry system 101 Jun 5 10:03 file1
$
```

You have given your group and all other system users write permission to `file1`.

5.4.1.2 Changing Directory Permissions

The procedure for changing directory permissions is the same as that for changing file permissions. However, to list the information about a directory, you use the `ls -ld` command:

```
$ ls -ld project
drwxr-xr-x 2 larry system 32 Jun 5 10:07 project
$
```

Now change the permissions with the `chmod g+w` command so that the group (`g`) has write permission (`+w`) for the directory `project`:

```
$ chmod g+w project
$ ls -ld project
drwxrwxr-x 2 larry system 32 Jun 5 10:07 project
$
```

5.4.1.3 Using Pattern-Matching Characters

If you want to make the same change to the permissions of all entries in a directory, you can use the pattern-matching character asterisk (`*`) with the `chmod` command. For information on pattern-matching characters, see Chapter 2.

In the following example, the command `chmod g+x *` gives execute (`x`) permission to the group (`g`) for all files (`*`) in the current directory:

```
$ chmod g+x *
$
```

Now enter the `ls -l` command to show that the group now has execute (x) permission for all files in the current directory:

```
$ ls -l
total 7
-rw-rwxrw- 1 larry system 101 Jun 5 10:03 file1
-rw-r-xr-- 1 larry system 171 Jun 5 10:03 file2
-rw-r-xr-- 1 larry system 130 Jun 5 10:06 file3
drwxrwxr-x 2 larry system 32 Jun 5 10:07 project
-rw-r-xr-- 1 larry system 0 Jun 5 11:03 record1
-rw-r-xr-- 1 larry system 0 Jun 5 11:03 record6
drwxr-xr-x 2 larry system 32 Jun 5 10:31 reports
$
```

5.4.1.4 Setting Absolute Permissions

An absolute permission assignment (=) resets all permissions for a file or files, regardless of how the permissions were set previously.

In the following example, the `ls -l` command lists the permissions for the `file3` file. Then the `chmod a=rwx` command gives all three permissions (rwx) to all users (a).

```
$ ls -l file3
-rw-r-x-r-- 1 larry system 130 Jun 5 10:06 file3
$ chmod a=rwx file3
$ ls -l file3
-rwxrwxrwx 1 larry system 130 Jun 5 10:06 file3
$
```

You can also use an absolute assignment to remove permissions. In the following example, the `chmod a=rw newfile` command removes the execute permission (x) for all groups (a) from the `file3` file:

```
$ chmod a=rw file3
$ ls -l file3
-rw-rw-rw- 1 larry system 130 Jun 5 10:06 file3
$
```

5.4.2 Specifying Permissions with Octal Numbers

You can also use octal numbers to change file and directory permissions.

To use octal number permission codes with the `chmod` command, enter the command in the following form:

```
chmod octalnumber filename
```

The *octalnumber* entry is a 3-digit octal number that specifies the permissions for owner, group, and others. The *filename* entry is the name

of the file whose permissions you want to change. It can be the name of the file or a list of filenames separated by spaces. You may also use pattern-matching characters to specify files. See Section 5.4.1.3 for more information.

An octal number corresponds to each type of permission:

4 = read
2 = write
1 = execute

To specify a group of permissions (a permissions field), add together the appropriate octal numbers (r, w, and x denote read, write, and execute respectively):

3 = -wx (2 + 1)
6 = rw- (4 + 2)
7 = rwx (4 + 2 + 1)
0 = --- (no permissions)

Table 5-2 lists the eight possible permission combinations.

Table 5-2: Permission Combinations

Octal Number	Binary Number	Permissions	Description
0	000	None	No permissions granted
1	001	--x	Execute
2	010	-w-	Write
3	011	-wx	Write/execute
4	100	r--	Read
5	101	r-x	Read/execute
6	110	rw-	Read/write
7	111	rwx	Read/write/execute

The entire permission code for a file or directory is specified with a 3-digit octal number, one digit each for owner, group, and others. Table 5-3 shows some typical permission codes and how they relate to the permission fields.

Table 5-3: How Octal Numbers Relate to Permission Fields

Octal Number	Owner Field	Group Field	Others Field	Complete Code
777	rwX	rwX	rwX	rwXrwXrwX
755	rwX	r-X	r-X	rwXr-Xr-X
700	rwX	---	---	rwX-----
666	rw-	rw-	rw-	rw-rw-rw-

For example, you could use the following commands to change the permission of `file3` using octal numbers:

```
$ ls -l file3
-rw-rw-rw- 1 larry system 130 Jun 5 10:06 file3
$ chmod 754 file3
$ ls -l file3
-rwxr-xr-- 1 larry system 130 Jun 5 10:06 file3
$
```

5.5 Setting Default Permissions with the User Mask (umask)

Every time you create a file or a directory, default permissions are established for it. These default permissions are initially established either by the operating system or the program you are running (both will be considered to be the creating program in the `umask` description that follows). Setting default permissions relieves you from the task of specifying permission codes explicitly every time you create a file or directory. The operating system assigns the default permission values of `777` for executable files and `666` for all other files.

If you want to further restrict whatever permissions are established by a program when it creates a file or directory, you must specify a user mask with the `umask` command.

The user mask is a numeric value that determines the access permissions when a file or directory is created. As a result, when you create a file or directory, its permissions are set to what the creating program specifies, *minus* what the `umask` value forbids.

The `umask` command has the following format:

```
umask octalnumber
```

The *octalnumber* entry is a 3-digit octal number that specifies the permissions to be subtracted from the default permissions (777 or 666).

Setting the user mask is very similar to setting the permission bits discussed in Section 5.4.2. The permission code for a file or directory is specified with a 3-digit octal number. Each digit represents a type of permission. The position of each digit (first, second, or third) represents 3 bits that correspond to the following:

- The first digit is for the `owner` of the file (you).
- The second digit is for the `group` of the file.
- The third digit is for `others`.

When you set the `umask`, you are actually specifying which permissions are *not* to be granted regardless of the permissions requested by the file creating program.

Table 5-4 lists the eight possible `umask` permission combinations for easy reference. Note that the `umask` permission values are the *inverse* of those specified for regular permission codes. Also note that these permission values are applied to those set by the creating program.

Table 5-4: The `umask` Permission Combinations

Octal Number	Allowed Permissions	Description
0	<code>rxw</code>	Read/write/execute
1	<code>rw-</code>	Read/write
2	<code>r-x</code>	Read/execute
3	<code>r--</code>	Read
4	<code>-wx</code>	Write/execute
5	<code>-w-</code>	Write
6	<code>--x</code>	Execute
7	<code>none</code>	No permissions granted

For example, if you specify a user mask of 027 (and the file is executable):

- The `owner` is allowed all permissions requested by the program creating the file.
- The `group` is *not* allowed write permission.

- The `others` are *not* allowed any permissions.

A good user mask value to set for your own files and directories depends upon how freely information resources are shared on your system. The following guidelines may be useful:

- In a very open computing environment, you might specify `000` as a user mask value, which allows no restrictions on file/directory access. As a result, when a program creates a file and specifies permission codes for it, the user mask imposes no restrictions on what the creating program has specified.
- In a more secure computing environment, you might specify `066` as a user mask value, which allows you total access, but prevents all others from being able to read or write to your files. As a result, when a file is created, its permissions are set to what the creating program specifies, *minus* the user mask restrictions that prevent read/write access for everyone but you.
- In a very secure computing environment, you might specify `077` as a user mask value, which means that only you have access to your files. As a result, when a file is created, its permissions are set to what the creating program specifies, *minus* the user mask restrictions that prevent anyone else from reading, writing, or executing your files.

To show you how `umask` works, assume that you have entered the following command:

```
$ umask 037
```

This command establishes a permission code of `740` (if the file is executable) and produces the following results:

- You (the owner) are allowed all permissions.
- Members of your group are not allowed write and execute permissions.
- All others are not allowed any permissions.

Further, assume that you have just created a file. By default, your editor always assigns the following default permissions: owners are allowed all permissions, and all others only read and execute permissions. However, since you have previously set a user mask of `037`, it further restricts the file permissions. As a result, the owner still has all permissions, but the group cannot execute the file, and all others have no permissions.

5.5.1 Setting the `umask`

You may activate the `umask` command in two ways:

- Include the `umask` command in your login script. This is the most common and efficient way to specify your user mask because the specified value is set automatically for you whenever you log in. For a discussion of login scripts, see Chapter 7. For examples of `umask` commands in login scripts, see Chapter 8.
- Enter the `umask` command at the shell prompt during a login session. The user mask value you set is in effect for that login session only.

For a more detailed example of how the user mask works in restricting permissions for files you create with a text editor, follow the steps in this procedure:

1. Enter the following command to find out what the current value of your user mask is:

```
$ umask
```

If the user mask value is 000, there are no restrictions on the permissions established by file-creating programs. Go to step 3.

If the user mask value is set, write it down. Go to step 2.

2. Set the user mask value to 000 so that there will be no restrictions on the permissions established by file-creating programs. Before resetting the user mask, make sure you have written down the current value in case you need to reset it.

Enter the following command:

```
$ umask 000
```

3. Create a file, save it, and then exit your editor.
4. Display the permissions of the file by using the `ls -l` command. We will assume for the sake of the example that read/write permissions are granted for all users:

```
$ ls -l
-rw-rw-rw- 1 user-name 15 Oct 27 14:42 yourfile
$
```

5. Reset the user mask to 022 by entering the following command:

```
$ umask 022
```

A user mask of 022 establishes the following permission restrictions: owners are allowed all permissions and all others are allowed only read and execute permissions.

6. Create another file, save it, and then exit your editor.

7. Display the permissions of the file by entering the `ls -l` command:

```
$ ls -l
-rw-r--r-- 1 user-name 15 Oct 27 14:45 yourfile2
$
```

Notice that the write permissions for the group and all others have been removed in accordance with the user mask value of 022.

8. Reset the user mask to its original value or to another value (if you choose).

Note

A user with superuser privileges can override whatever access restrictions you impose on files and directories. For more information on this topic, see Section 5.7.

On occasion, the results you obtain when specifying a user mask may vary from what you intended. If so, see your system administrator.

The operating system provides a default user mask value of 022, which allows the owner all permissions, but prevents members of your group or any other users from writing to your files. However, your system's user mask default may vary.

5.6 Changing Your Identity to Access Files

The `su` command allows you to alter your identity during a login session. A reason for altering your identity is to be able to access files that you do not own. To protect system security, you should not assume another identity without the owner's or the system administrator's permission.

The `su` command allows you to log in to another user's account only if you know that user's password. The `su` command authenticates you and then resets both the process's user ID and the **effective user ID** to the value of the newly specified user ID. The effective user ID is the user ID currently in effect for the process, although it may not be the user ID of the person logged in.

The format of the `su` command is:

```
su username
```

The *username* entry is the username of the person whose identity you want to assume.

If after altering your identity, you want to confirm what identify you have assumed, use the `whoami` command. This command displays the username of the identity you have assumed.

After completing your work under a new identity, you should return to your own login identity. To do so, press Ctrl/D or enter the `exit` command.

The following example shows how Juan assumes Lucy's identity with the `su` command, confirms it with the `whoami` command, removes a file, and then returns to his own login identity with the `exit` command:

```
$ su lucy
Password: ...
$ whoami
lucy
$ rm file9
$ exit
$ whoami
juan
$
```

For more information, see the `su(1)` and `whoami(1)` reference pages.

5.7 Superuser Concepts

Every system has a superuser who has permissions that supersede those of ordinary users. This superuser is often referred to as root.

The root user has absolute power over the running of the system. This user has access to all files and all devices and can make any changes to the system. The root user is said to have superuser privileges.

The following is a list of tasks ordinarily performed by root users:

- Edit files not normally changeable by ordinary users (for example, `/etc/passwd`)
- Be able to change ownership and permissions of all files
- Execute restricted commands like `mount` or `reboot`
- Kill any process running on your system
- Add and remove user accounts
- Boot and shut down the system
- Back up the system

Many of the preceding tasks are performed by system administrators who require superuser privileges. The system administrator's job is to manage the system by performing the preceding tasks, installing new software, analyzing system performance, and reporting hardware failures.

Depending upon your computing environment, you may or may not be the system administrator for your system or have root privileges. Your site configuration as well as your job responsibilities will determine your privileges.

If you work from a terminal or workstation that accesses a centralized system, you will probably not be the system administrator or have root privileges. In this situation, the system administrator, who is in charge of maintaining, configuring, and upgrading the system, will be the person who has root privileges.

If you perform your tasks from a workstation that is either independent or networked to other workstations or systems, you may indeed have root privileges for your own workstation, but not be the system administrator of your site. In this situation, you would maintain your own workstation only. However, the system administrator would still maintain shared machines and networks.

To become a root user, use the `su` command. You must also know the password for the root user. The format of the `su` command is:

su root

The following example shows how Juan becomes a root user to perform an administrative task:

```
$ su root
Password: ...
# _
```

The new prompt, a number sign (#), indicates that Juan has become a root user and that a shell has been created for his use. The root user shell (often the C shell) is defined in the `/etc/passwd` file. Juan may now perform the administrative task.

Caution

Because the root user has absolute power over the system, the password should be carefully protected. Otherwise, unauthorized use of the system may result in corruption or destruction of data.

After completing your work as the root user, you should return to your own login identity. To do so, press `Ctrl/D` or enter the `exit` command. You are then returned to the system prompt.

5.8 Changing Owners and Groups (`chown` and `chgrp`)

In addition to setting permissions, you can control how a file or directory is used by changing its owner or group. Use the `chown` command to change the owner and the `chgrp` command to change the group.

Note

In order to use the `chown` command, you must have superuser privileges. For more information on this topic, see Section 5.7.

Enter the `chown` command in the following form:

chown *owner filename*

The *owner* entry is the username of the new owner of the file. The *filename* entry is a list of one or more files whose ownership you want to change. You may also use pattern-matching characters to specify files. See Section 5.4.1.3 for more information.

Enter the `chgrp` command in the following form:

chgrp *group file*

The *group* entry is the group ID or group name of the new group. Note that to change the group ownership of a file, you must be a member of the group to which you are changing the file. The *file* entry is a list of one or more files whose ownership you want to change.

For more information, see the `chown(1)` and `chgrp(1)` reference pages.

5.9 Additional Security Considerations

The security guidelines enforced at your site protect your files from unauthorized access. See your system administrator for complete information about security guidelines.

In addition, it is wise to avoid running untrusted software (software that is from an unknown source or that has not been validated for system security). When you run a program, that program has all of your access rights, and nothing prevents the program from being used to illicitly access, observe, or alter sensitive files.

You should be aware of three types of programs that compromise security:

- Trojan horse

A trojan horse is a program that performs, or appears to perform, its defined task properly; however, it also performs hidden functions that may be malicious. A trojan horse program emulates the program that you intended to run, but may perform an unwanted action. It might vandalize your files by altering or deleting them, or compromise the files by making illegal copies of them.

A typical trojan horse is the `login` trojan horse, which mimics the system's login prompt on the display and waits for you to enter a username and password. The program mails or copies this information to the user responsible for the trojan horse. As the trojan horse exits, it

displays `Login incorrect`. The real `login` program then runs. Most users assume they typed the password incorrectly, and are unaware that they were deceived.

- Computer worm

A computer worm is a program that moves around a computer network, making copies of itself. For example, a login computer worm can log in to a system, copy itself onto the system, start running, log in to another system, and then continue this process indefinitely.

- Computer virus

A computer virus program is really a type of trojan horse. Normally, a trojan horse waits passively for the right user to run it (usually a privileged user). Viruses spread by inserting themselves in other executable files, thus increasing the threat and extent of compromise of privacy or integrity.

Be careful of programs that were not installed by the person who administers your system. Programs that are obtained from bulletin boards and other unknown origins are particularly suspect. Even if the program includes source code, it is not always possible to examine the program carefully enough to determine if it is trustworthy.

Using Processes **6**

This chapter explains the operating system processes. After completing this chapter, you will be able to:

- Understand programs and processes
- Redirect process input, output, and errors
- Run processes in the foreground and background
- Check the status of processes
- Cancel processes
- Display information about users and their processes

A good way to learn about the preceding topics is to try the examples in this chapter. You should do each example so that the information on your screen is consistent with the information in this book.

6.1 Understanding Programs and Processes

A **program** is a set of instructions that a computer can interpret and run. You may think of most programs as belonging to one of two categories:

- Application programs such as text editors, accounting packages, or electronic spreadsheets
- Programs that are components of the operating system such as commands, the shell (or shells), and your login procedure

While a program is running, it is called a **process**. The operating system assigns every process a unique number known as a **process identifier**.

The operating system can run a number of different processes at the same time. When more than one process is running, a scheduler built into the operating system gives each process its fair share of the computer's time, based on established priorities.

6.2 Understanding Standard Input, Output, and Error

When a process begins executing, the operating system opens three files for the process: `stdin` (standard input), `stdout` (standard output), and `stderr` (standard error). Programs use these files as follows:

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

In most cases, the default standard input, output, and error mechanisms will serve you well. However, there are times when it is useful to redirect the standard input, output, and error. The following sections describe these procedures.

6.2.1 Redirecting Input and Output

A command usually reads its input from the keyboard (standard input) and writes its output to the display (standard output). You may want a command to read its input from a file, write its output to a file, or both. You can select input and output files for a command with the shell notation shown in Table 6-1. This notation can be used in all shells.

Table 6-1: Shell Notation for Reading Input and Redirecting Output

Notation	Action	Example
<	Reads standard input from a file	<code>wc <file3</code>
>	Writes standard output to a file	<code>ls >file3</code>
>>	Appends (adds) standard output to the end of a file	<code>ls >>file3</code>

The following sections explain how to read input from a file and how to write output to a file.

6.2.1.1 Reading Input from a File

All shells allow you to redirect the standard input of a process so that input is read from a file, instead of from the keyboard.

You can use input redirection with any command that accepts input from `stdin` (your keyboard). You cannot use input redirection with commands, such as `who`, that do not accept input.

To redirect input, use the left-angle bracket (<), as the following example shows:

```
$ wc <file3
   3      27      129
$
```

The `wc` (word count) command counts the number of lines, words, and bytes in the named file. So `file3` contains 3 lines, 27 words, and 129 bytes. If you do not supply an argument, the `wc` command reads its input from the keyboard. In this example, however, input for `wc` comes from the file named `file3`.

Note that in the preceding example, you could have entered the following, and displayed the same output:

```
wc file3
```

This is because most commands allow the input file to be specified without the left-angle bracket (<).

However, there are a few commands like `mail` that require the use of the left-angle bracket (<) for special functions. For example, note the following command:

```
$ mail juan <report
```

This command mails to the user `juan` the file `report`. For more information about `mail`, see the `mail(1)` reference page.

6.2.1.2 Redirecting Output

All shells allow you to redirect the standard output of a process from the screen (the default) to a file. As a result, you can store the text generated by a command into a new or existing file.

To send output to a file, use either a right-angle bracket (>) or two right-angle brackets (>>).

The right-angle bracket (>) causes the shell to:

- Replace the contents of the file with the output of the command, if the file exists.
- Create the file, if the file does not exist and place the output of the command into the file.

Two right-angle brackets (>>) add (append) the output of the command to the end of a file that exists. If you use two right-angle brackets (>>) to write output to a file that does not exist, the shell creates the file containing the output of the command.

In the next example, the output of `ls` goes to the file named `file`:

```
$ ls >file
$
```

If the file already exists, the shell replaces its contents with the output of `ls`. If `file` does not exist, the shell creates it.

In the following example, the shell adds the output of `ls` to the end of the file named `file`:

```
$ ls >>file
$
```

If `file` does not exist, the shell creates it.

In addition to their standard output, processes often produce error or status messages known as diagnostic output. For information about redirecting diagnostic output, see the following section.

6.2.2 Redirecting Standard Error to a File

When a command executes successfully, it displays the results on the standard output. When a command executes unsuccessfully, it displays error messages on the default standard error file, the screen. However, the shell allows you to redirect the standard error of a process from the screen to a file.

Redirection symbols and syntax vary among shells. The following sections describe standard error redirection for the Bourne, Korn, and C shells.

6.2.2.1 Bourne and Korn Shell Error Redirection

The general format for Bourne and Korn shell standard error redirection is the following:

```
command 2> errorfile
```

The *command* entry is an operating system command. The *errorfile* entry is the name of the file to which the process writes the standard error. The `2>` is a file descriptor digit combined with the right-angle bracket (`>`), the output redirection symbol. The file descriptor digit tells the shell what standard file to access so that its contents may be redirected. The file descriptor digit 2 indicates that the standard error file is being redirected.

In fact, for the Bourne and Korn shells, a file descriptor digit is associated with each of the files a command ordinarily uses:

- File descriptor 0 (same as the left-angle bracket [`<`]) specifies standard input (the keyboard).
- File descriptor 1 (same as the right-angle bracket [`>`]) specifies standard output (the screen).

- File descriptor 2 specifies standard error (screen).

In the following example, an error is redirected to the `error` file when the `ls` command attempts to display the nonexistent file, `reportx`. The contents of `error` file are then displayed:

```
$ ls reportx 2> error
$ cat error
reportx not found
$
```

Although only standard error is redirected to a file in the preceding example, typically you would redirect both standard error and standard output. See Section 6.2.3 for more information.

For many commands, the difference between standard output and standard error is difficult to see. For instance, if you use the `ls` command to display a nonexistent file, an error message displays on the screen. If you redirect the error message to a file as in the previous example, the output is identical.

6.2.2.2 C Shell Error Redirection

The general format for C shell standard error redirection is the following:

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

The *command* entry is any operating system command. The *outfile* entry is the name of the file to which the process writes the standard output. The right-angle bracket (>) redirects the standard error to a file. The *errorfile* entry is the name of the file to which the process writes the standard error. Note that in this command format, the parentheses are mandatory.

6.2.3 Redirecting Both Standard Error and Standard Output

In the preceding sections, you learned how to redirect standard output and standard error separately. Usually, however, you would redirect both standard output and standard error at the same time. Standard output and standard error can be written to different files or to the same file.

For the Bourne and Korn shells, the general format for redirecting both standard output and standard error to different files is the following:

```
command>outfile2>errorfile
```

The *command* entry is an operating system command. The *outfile* entry is the file to which the process writes the standard output. The `2>` symbol redirects the error output. The *errorfile* entry is the file where the process writes the standard error.

For the C shell, the general format for redirecting both standard output and standard error to different files is the following:

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

The *command* entry is an operating system command. The *outfile* entry is the file to which the process writes the standard output. The right-angle bracket and ampersand (>&) symbol redirects the error output. The *errorfile* entry is the file where the process writes the standard error. Note that in this command format, the parentheses are mandatory. See Section 6.2.2.2 for more information.

For the Bourne and Korn shells, the general format for redirecting both standard output and standard error to the same file is the following:

```
command1>outfile2>&1
```

The *command* entry is an operating system command. The 1> symbol redirects the standard output. The *outfile* entry is the file to which the process writes the standard output. The 2>&1 symbol tells the shell to write the standard error (file descriptor 2) in the file associated with the standard output (>&1), *outfile*.

For the C shell, the general format for redirecting both standard output and standard error to the same file is the following:

```
command>&outfile
```

The *command* entry is an operating system command. The *outfile* entry is the file to which the process writes the standard output. The right-angle bracket and ampersand (>&) symbol tells the shell to write the standard output and standard error to the same file specified by *outfile*.

6.3 Running Several Processes Simultaneously

The operating system can run a number of different processes at the same time. This capability makes it a **multitasking** operating system, which means that the processes of several users can run at the same time.

These different processes can be from one or multiple users. As a result, you do not have to enter commands one at a time at the shell prompt. Instead, you can run both foreground and background processes simultaneously. The following sections describe both foreground and background processes.

6.3.1 Running Foreground Processes

Normally, when you enter a command on the command line, you wait for the results to display on your screen. Commands entered singly at the shell prompt are called **foreground processes**.

Most commands take a short time to execute – perhaps a second or two. However, some commands require longer execution times. If a long-duration command runs as a foreground process, you cannot execute other commands until the current one finishes. As a result, you may want to run a long-duration command as a background process. The following section describes background processes.

6.3.2 Running Background Processes

Generally, **background processes** are most useful with commands that take a long time to run. Instead of tying up your screen by entering a long-duration command as a foreground process, you can execute a command as a background process. You can then continue with other work in the foreground.

To run a background process, you end the command with an ampersand (&). Once a process is running in the background, you can perform additional tasks by entering other commands at your workstation.

After you create a background process, the following takes place:

- The Process Identification Number (PID) is displayed. The operating system creates and assigns PIDs so that all processes currently running on the system can be tracked. (In the Korn or the C shell, job numbers are assigned as well.)
- The prompt returns so that you can enter another command.
- In the C shell, a message is displayed when the background process is complete.

When you create a background process, note its PID number. The PID number helps you to monitor or terminate the process. See Section 6.4 for more information.

Because background processes increase the total amount of work the system is doing, they may also slow down the rest of the system. This may or may not be a problem, depending upon how much the system slows and the nature of the other work you or others do while background processes run.

Most processes direct their output to standard output, even when they run in the background. Unless redirected, standard output goes to your workstation. Because the output from a background process may interfere with your other work on the system, it is usually good practice to redirect the output of a background process to a file or to a printer. Then you can look at the output whenever you are ready. For more information about redirecting output, see the examples later in this chapter as well as Section 6.2.

The examples in the remainder of this chapter use a command that takes more than a few seconds to run:

```
$ find / -type f -print
```

This command displays the pathnames for all files on your system. You do not need to study the `find` command in order to complete this chapter – it is used here simply to demonstrate how to work with processes. However, if you want to learn more about the `find` command, see the `find(1)` reference page.

In the following example, the `find` command runs in the background (by entering an ampersand [`&`]) and redirects its output to a file named `dir.paths` (by using the right-angle bracket [`>`] operator):

```
$ find / -type f -print >dir.paths &
24
$
```

When the background process starts, the system assigns it a PID (Process Identification) number (24 in this example), displays it, and then prompts you for another command. Your process number will be different from the one shown in this and following examples.

If you use the Korn or C shell, job numbers are assigned as well. In the C shell, the preceding example looks like this:

```
%find / -type f -print >dir.paths &
[1] 24
% _
```

Note that the job number [1] is displayed to the left of the PID number.

You can check the status of the process with the `ps` (process status) or the `jobs` command (Korn and C shell). You can also terminate a process with the `kill` command. See the following section for more information about these commands.

In the C shell, when the background process is completed, a message is displayed:

```
[1] 24 Done find / -type f -print >dir.paths
```

The completion message displays the job number and the PID, the status Done, and the command executed.

6.4 Monitoring and Terminating Processes

Use the `ps` (process status) command to find out which processes are running and to display information about those processes. In the Korn and C shells, you also can use the `jobs` command to monitor background processes.

If you need to stop a process before it is finished, use the `kill` command.

The following sections describe how to monitor and terminate processes.

6.4.1 Checking Process Status

The `ps` command allows you to monitor the status of all active processes, both foreground and background. In the Korn and C shell, you also can use the `jobs` command to monitor background processes only. The following sections describe the `ps` and the `jobs` command.

6.4.1.1 The `ps` Command

The `ps` command has the following form:

ps

In the following example, the `ps` command displays the status of all processes associated with your workstation under the following headings:

```
$ ps
PID          TT    STAT      TIME    COMMAND
29670        p4    I         0:00.00  -sh (csh)
   515        p5    S         0:00.00  -sh (csh)
28476        p5    R         0:00.00  ps
   790        p6    I         0:00.00  -sh (csh)
$
```

You interpret the display under these entry headings as follows:

PID	Process identification. The system assigns a process identification number (PID number) to each process when that process starts. There is no relationship between a process and a particular PID number; that is, if you start the same process several times, it will have a different PID number each time.
TT	Controlling terminal device name. On a system with more than one workstation, this field tells you which workstation started the process. On a system with only one workstation, this field can contain the designation <code>console</code> or the designation for one or more virtual terminals.
STAT	Symbolic process status. The system displays the state of the process, with a sequence of up to four alphanumeric characters. For more information, see the <code>ps(1)</code> reference page.
TIME	Time devoted to this process by the computer is displayed in minutes, seconds, and hundredths of seconds starting when you enter <code>ps</code> .
COMMAND	The name of the command (or program) that started the process.

You can also check the status of a particular process by using the `-p` flag and the PID number with the `ps` command. The general format for checking the status of a particular process is the following:

```
ps -p PIDnumber
```

The `ps` command also displays the status of background processes. If there are any background processes running, they will be displayed along with the foreground processes. The following example shows how to start a `find` background process and then check its status:

```
$ find / -type f -print >dir.paths &
25
$ ps -p25
PID TTY          TIME COMMAND
 25  console 0:40:00  find
$
```

You can check background process status as often as you like while the process runs. In the following example, the `ps` command displays the status of the preceding `find` process five times:

```
$ ps -p25
PID TTY          TIME COMMAND
 25  console 0:18:00  find
$ ps -p25
PID TTY          TIME COMMAND
 25  console 0:29:00  find
$ ps -p25
PID TTY          TIME COMMAND
 25  console 0:49:00  find
$ ps -p25
PID TTY          TIME COMMAND
 25  console 0:58:00  find
$ ps -p25
PID TTY          TIME COMMAND
 25  console 1:02:00  find
$ ps -p25
PID TTY          TIME COMMAND
$
```

Notice that the sixth `ps` command returns no status information because the `find` process ended before the last `ps` command was entered.

Generally, the simple `ps` command described here tells you all you need to know about processes. However, you can control the type of information that the `ps` command displays by using more of its flags. One of the most useful `ps` flags is `-e`, which causes `ps` to return information about all processes, *not* just those associated with your terminal or workstation. For an explanation of all `ps` command flags, see the `ps(1)` reference page.

6.4.1.2 The jobs Command

The Korn shell and the C shell display both a job and a PID when a background process is created. The `jobs` command reports the status of all background processes only, based upon the job number.

The `jobs` command has the following form:

jobs

Adding the `-l` flag displays both the job number and the PID.

The following example shows how to start a `find` process and then check its status in the C shell with the `jobs -l` command:

```
% find / -type f -print >dir.paths &
[2] 26
% jobs -l
[2] +26 Running find / -type f -print >dir.paths &
% _
```

The status message displays both the job ([2]) and the PID number (26), the status `Running`, and the command executed.

6.4.2 Canceling a Foreground Process (Ctrl/C)

To cancel a foreground process (stop an executing command), press `Ctrl/C`. The command stops executing, and the system displays the shell prompt. Note that canceling a foreground process is the same as stopping command execution (described in Chapter 1).

Most simple operating system commands are not good examples for demonstrating how to cancel a process – they run so quickly that they finish before you have time to cancel them. However, the following `find` command runs long enough for you to cancel it (after the process runs for a few seconds, you can cancel it by pressing `Ctrl/C`):

```
$ find / -type f -print
/usr/sbin/acct/acctcms
/usr/sbin/acct/acctcoN1
/usr/sbin/acct/acctcon2
/usr/sbin/acct/acctdisk
/usr/sbin/acct/acctmrg
/usr/sbin/acct/accton
/usr/sbin/acct/acctprc1
/usr/sbin/acct/acctprc2
/usr/sbin/acct/acctwtmp
/usr/sbin/acct/chargefee
/usr/sbin/acct/ckpacct
/usr/sbin/acct/dodisk
Ctrl/C
$
```

The system returns the shell prompt to the screen. Now you can enter another command.

6.4.3 Canceling a Background Process (kill)

If you decide, after starting a background process, that you do not want the process to finish, you can cancel the process with the `kill` command. Before you can cancel a background process, however, you must know its PID number.

If you have forgotten the PID number of that process, you can use the `ps` command to list the PID numbers of all processes. If you are a C or Korn Shell user, it is more efficient to use the `jobs` command to list background processes only.

The general format for terminating a particular process is the following:

kill *PIDnumber*

If you want to end all the processes you have started since login, use the `kill 0` command. You do not have to know the PID numbers to use `kill 0`. Because this command deletes all of your processes, use this command carefully.

The following example shows how to start another `find` process, check its status, and then terminate it:

```
$ find / -type f -print >dir.paths &
38
$ ps
PID  TT  STAT      TIME    COMMAND
520  p4  I        0:11:10  sh
38   p5  I        0:10:33  find
1216 p6  S        0:01:14  qdaemon
839  p7  R        0:03:55  ps
$ kill 38
$ ps
38 Terminated
PID  TT  STAT      TIME    COMMAND
520  p4  I        0:11:35  sh
1216 p6  S        0:01:11  qdaemon
839  p7  R        0:03:27  ps
$
```

The command `kill 38` stops the background `find` process, and the second `ps` command returns no status information about PID number 38. The system does not display the termination message until you enter your next command.

Note that in this example, `kill 38` and `kill 0` have the same effect because only one process was started from this terminal or workstation.

In the C shell, the `kill` command has the following format:

```
kill % jobnumber
```

The following example uses the C shell to start another `find` process, to check its status with the `jobs` command, and then to terminate it:

```
% find / -type f -print >dir.paths &
[3] 40
% jobs -l
[3] +40 Running  find / -type f -print >dir.paths &
% kill %3
% jobs -l
[3] +Terminated  find / -type f -print > dir.paths
% _
```

6.4.4 Suspending and Resuming a Foreground Process (C Shell Only)

Stopping a foreground process and resuming it can be helpful when you have a long-duration process absorbing system resources and you need to do something quickly.

Rather than waiting for process completion, you can stop the process temporarily (suspend it), perform your more critical task, and then resume the process. Suspending a process is available for C shell users only.

To suspend a process, press `Ctrl/Z`. A message will display listing the job number, the status suspended, and the command executed.

Once you are ready to resume the process, enter:

```
% n
```

To resume the process in the background, enter:

```
% n &
```

where the *n* entry is the number of the stopped job.

The following example starts a `find` process, suspends it, checks its status, resumes it, and then terminates it:

```
% find / -type f -print >dir.paths &
[4] 41
% jobs -l
[4] +41 Running  find / -type f -print >dir.paths &
% Ctrl/Z
Suspended
% jobs -l
[4] +Stopped    find / -type f -print > dir.paths
% 4 &
[4] find / -type f -print >dir.paths &
% kill %4
[4] +Terminated  find / -type f -print > dir.paths
```

Once a process is suspended, you may also resume it by entering the `fg` command. If a currently running process is taking too long to run and is tying up your keyboard, you can use the `bg` command to place the process in the background and enter other commands.

The following example starts a `find` process, suspends it, puts the process in the background, copies a file, and then resumes the process in the foreground:

```
% find / -type f -print >dir.paths
Ctrl/Z
Suspended
% bg
[5] find / -type f -print > dir.paths &
% cp salary1 salary2
% fg
find / -type f -print > dir.paths
%
```

6.5 Displaying Information About Users and Their Processes

The operating system provides the following commands that can tell you who is using the system and what they are doing:

- `who` Displays information about currently logged in users.
- `w` Displays information about currently logged in users and what they are currently running on their workstations.
- `ps au` Displays information about currently logged in users and the processes they are running.

The `who` command allows you to determine who is logged into the system. It may be especially useful, for example, when you want to send a message and want to know whether the person is currently available.

In the following example, information about all currently logged in users is displayed:

```
$ who
juan tty01 Jan 15 08:33
chang tty05 Jan 15 08:45
larry tty07 Jan 15 08:55
tony tty09 Jan 15 07:53
lucy pts/2 Jan 15 11:24 (boston)
$
```

Note that the `who` command lists the username of each user on the system, the workstation being used, and when the person logged in. In addition, if a user is logged in from a remote system, the name of the system is listed. For

example, `lucy` logged in remotely from the system `boston` on Jan 15 at 11:24.

The actual display format of `who` varies according to your current locale. See Appendix C for more information about locales.

The `who -u` command gives all the information of the `who` command and also displays the PID of each user and the number of hours and minutes since there was activity at a workstation. Activity for less than a minute is indicated by a dot (.).

In the following example, all currently logged in users are displayed:

```
$ who -u
juan  tty01  Jan 15  08:33  01:02  50
chang tty05  Jan 15  08:45  .      52
larry tty07  Jan 15  08:55  .      58
tony  tty09  Jan 15  07:53  01:20  60
lucy  pts/5  Jan 15  11:24  .      65 (boston)
$
```

Note that in the preceding example, `juan` and `tony` have been inactive for over an hour, while `chang`, `larry`, and `lucy` have been inactive for less than a minute.

Now that you know how to find out who is active on your system, you may want to find out what command each person is currently executing. The `w` command displays the command that is currently running at each user's workstation.

In the following example, information about all users (the `User` column) and their current commands (the `what` column) is displayed:

```
$ w
11:02 up 3 days, 2:40, 5 users, load average: 0.32, 0.20, 0.00
User  tty      login@  idle   JCPU   PCPU   what
juan  tty01    8:33am  12     54     14    -csh
chang tty05    8:45am          6:20   26    mail
larry tty07    8:55          1:58    8    -csh
tony  tty09    7:53    3:10   22     4    mail
lucy  tty02    11:24   1:40   18     4    -csh
$
```

In addition, the `w` command displays the following information:

- The `tty` column: user's workstation
- The `login@` column: user's login time
- The `idle` column: amount of time since the user entered a command
- The `JCPU` column: total CPU time used during the current login session
- The `PCPU` column: CPU time used by the command that is currently executing

On certain occasions, you may want to have a detailed listing of current processes (both foreground and background) and the users who are running them. To get such a listing, use the `ps au` command. In the following example, information about five users and their active processes is displayed:

```
$ ps au
USER      PID %CPU %MEM    SZ   RSS TT  STAT      TIME    COMMAND
juan    26300 16.5  0.8  441   327 p3  R       0:02:01   ps au
chang  25821  7.0  0.2  149    64 p4  R       0:12:23  mail -n
larry  25121  6.1  0.2  107    83 p22 R     26:25:07  tip modem
tony    11240  4.5  0.6  741   225 p19 R      1:57:46  emacs
lucy   26287  0.5  0.1   61    28 p1  S       0:00:00  more
$
```

The most important fields for the general user are the `USER`, `PID`, `TIME`, and `COMMAND` fields. For information on the remaining fields, see the `ps(1)` reference page.

Shell Overview **7**

This chapter introduces you to the operating system shells. After completing this chapter, you will be able to:

- Understand the purpose and general features of the Bourne, C, and Korn shells
- Change your shell
- Use command entry aids common to all shells
- Understand your shell environment as well as the role of login scripts, environment variables, and shell variables
- Set and clear environment and shell variables
- Understand how the shell finds commands on your system
- Write logout scripts
- Write and run basic shell procedures

This chapter covers features common to all operating system shells, with some descriptions of shell differences. For detailed information on specific Bourne, C, and Korn shell features, see Chapter 8.

7.1 Purpose of Shells

The user interfaces to the operating system are called shells. The shells are programs that interpret the commands you enter, run the programs you have asked for, and send the results to your screen.

The operating system provides the following shells:

- The Bourne shell (system default)
- The C shell
- The Korn shell

You may access any shell, depending upon the security restrictions in effect on your system as well as upon the licensing restrictions of the Korn shell. In any case, all shells perform the same basic function: they allow you to perform work on your system by executing commands.

In addition to interpreting commands, the shell can also be used as a programming language. You can create shell procedures that contain commands. Shell procedures are executed in the same way that you execute a program: on the command line after the shell prompt.

When you run a shell procedure, your current shell creates or **spawns** a subshell. A subshell is a new shell your current shell creates to run a program. Thus, any command the shell procedure executes (for example, `cd`) leaves the invoking shell unaffected.

Shell procedures provide a means of carrying out tedious commands, large or complicated sequences of commands, and routine or repetitive tasks.

See Section 7.10 for more information on shell programming.

7.2 Summary of Bourne, C, and Korn Shell Features

The operating system provides the following shells that have both command execution and programming capabilities:

- The Bourne shell (`sh`)

This is a simple shell that is easily used in programming. It is usually represented by a dollar sign (\$) prompt. This shell does not provide either the interactive features or the complex programming constructs (arrays and integer arithmetic) of the C shell or the Korn shell.

The Bourne shell also provides a restricted shell (`Rsh`). For more information, see Section 7.2.2.

- The C shell (`csh`)

This shell is designed for interactive use. It is usually represented by a percent sign (%) system prompt. The C shell provides some features for entering commands interactively:

- A command history buffer
- Command aliases
- Filename completion
- Command line editing

For more information on these features, see Section 7.2.1.

- The Korn shell (`ksh`)

This shell combines the ease of use of the C shell and the ease of programming of the Bourne shell. The system prompt is usually a dollar sign (\$) prompt. The Korn shell provides these features:

- The interactive features of the C shell

- The simple programming syntax of the Bourne shell
- Command line editing
- The fastest execution time
- Upward compatibility with the Bourne shell (that is, most Bourne shell programs will run under the Korn shell)

For more information on these features, see Section 7.2.1.

7.2.1 More Information on C and Korn Shell Features

Both the C and the Korn shells offer the following interactive features:

- Command history

The command history buffer stores the commands you enter and allows you to display them at any time. As a result, you can select a previous command, or parts of previous commands, and then reexecute them. This feature may save you time because it allows you to reuse long commands instead of retyping them. In the C shell, this feature requires some setup in the `.cshrc` file; in the Korn shell this feature is automatically provided.

- Command aliases

The command aliases feature allows you to abbreviate long command lines or rename commands. You do this by creating aliases for long command lines that you frequently use. For example, assume that you often need to move to the directory `/usr/chang/reports/status`. You could create an alias `status` that could move you to that directory whenever you enter `status` on the command line. In addition, aliases allow you to make up more descriptive names for operating system commands. For example, you could define an alias named `rename` for the `mv` command.

- Filename completion

In the C shell, the filename completion feature saves typing by allowing you to enter a portion of the filename. When you press the Escape key, the shell will complete the filename for you. See Section 8.2.4 for more information about filename completion in the C shell.

In the Korn shell, you can ask the shell to display a list of file names that match the partial name you entered. You may then choose among the displayed file names. See Section 8.4.5 for more information about filename completion in the Korn shell.

The Korn shell provides an inline editing feature that allows you to retrieve a previously entered command and edit it. To use this feature, you must know how to use a text editor such as `vi` or `emacs`.

For more information about these shell features, see Chapter 8.

7.2.2 The Restricted Bourne Shell

The operating system enhances system security by providing specified users a limited set of functions with a restricted version of the Bourne shell (`Rsh`). When these specified users log in to the system, they are given access to the restricted Bourne shell only. Your system administrator determines who has access to the restricted Bourne shell.

A restricted shell is useful for installations that require a more controlled shell environment. As a result, the system administrator can create user environments that have a limited set of privileges and capabilities. For example, all users that are guests to your system might be allowed access under the username `guest`. When logging in to your system, user `guest` would be assigned a restricted shell.

The actions of `Rsh` are identical to those of `sh`, except that the following actions are not allowed:

- Changing directories. The `cd` command is deactivated.
- Specifying pathnames or command names containing a slash (`/`).
- Setting the value of the `PATH` or the `SHELL` variables. For more information on these variables, see Section 7.5.2.
- Redirecting output with the right-angle brackets (`>` and `>>`).

For more detailed information on `Rsh`, see the `sh(1)` reference page. For information on how system administrators create restricted shells, see your system administrator.

7.3 Changing Your Shell

Whenever you log in, you are automatically placed in a shell specified by your system administrator. However, depending upon the security features in effect on your system, you can enter commands that will allow you to do the following:

- Determine which shell you are running
- Temporarily change your shell
- Permanently change your shell

The following sections describe these operations.

7.3.1 Determining What Shell You Are Running

To determine what shell you are currently running, enter the following command:

```
echo $SHELL
```

The filename of the shell you are running will display.

In the following example, assume that you are running the Bourne shell (sh):

```
$ echo $SHELL
/usr/bin/sh
$
```

Table 7-1 lists the filename that displays for each shell as well as the default system prompt (your system prompt may vary).

Table 7-1: Shell Names and Default Prompts

Shell	Shell Name	Default Prompt
Bourne	sh	\$
Restricted Bourne	Rsh	\$
C	csH	%
Korn	ksh	\$

7.3.2 Temporarily Changing Your Shell

You may experiment with using other shells if the security features on your system allow it.

To temporarily change your shell, enter the following command:

```
shellname
```

The *shellname* is the filename of the shell you want to use. See Table 7-1 for valid shell file names to enter on the command line. Once the shell is invoked, the correct shell prompt is displayed.

Once you are done using the new shell, you can return to your default shell by entering `exit` or by pressing `Ctrl/D`.

For example, assume that the Korn shell is your default shell. To change to the C shell and then back to the Korn shell, enter the following commands:

```
$ /usr/bin/csh
% exit
$
```

Note

If you are using the Bourne Restricted Shell, you cannot change to another shell.

7.3.3 Permanently Changing Your Shell

You may permanently change your default shell if the security features on your system allow it. If your current shell is the C shell, use the `chsh` command to change your default shell. If you don't use the C shell, change your default shell by contacting your system administrator.

In the C shell, enter the following command to change the default shell:

```
% chsh
Changing login shell for user.
Old shell: /usr/bin/csh
New shell:
```

Enter the name of the new shell. See Table 7-1 for valid shell names to enter on the command line.

After entering the `chsh` command, you must log out and log in again for the change to take effect.

7.4 Command Entry Aids

The following features of all operating system shells help you do your work:

- The ability to enter multiple commands and command lists
- Pipes and filters
- The ability to group commands
- Quoting

7.4.1 Using Multiple Commands and Command Lists

The shell usually takes the first word on a command line as the name of a command and then takes any other words as arguments to that command. The shell usually considers each command line as a single command. However, you can use the operators in Table 7-2 to execute multiple commands on a single command line.

Table 7-2: Multiple Command Operators

Operator	Action	Example
;	Causes commands to run in sequence.	<i>cmd1 ; cmd2</i>
&&	Runs the next command if the current command succeeds.	<i>cmd1 && cmd2</i>
	Runs the next command if the current command fails.	<i>cmd1 cmd2</i>
	Creates a pipeline.	<i>cmd1 cmd2</i>

The following sections describe running commands in sequence (;), running commands conditionally (|| and &&), and using pipelines (|).

7.4.1.1 Running Commands in Sequence with a Semicolon (;)

You can enter more than one command on a line if you separate commands with the semicolon (;).

In the following example, the shell runs `ls` and waits for it to finish. When `ls` is finished, the shell runs `who`, and so on through the last command:

```
$ ls ; who ; date ; pwd
change file3 newfile
amy console/1 Jun 4 14:41
Tue Jun 4 14:42:51 CDT 1996
/u/amy
$
```

If any one command fails, the others still execute successfully.

To make the command line easier to read, separate commands from the semicolon (;) with blanks or tabs. The shell ignores blanks and tabs used in this way.

7.4.1.2 Running Commands Conditionally

When you connect commands with two ampersand (&&) or pipe (|) operators, the shell runs the first command and then runs the remaining commands only under the following conditions:

- && The shell runs the next command only if the current command completes (a command indicates successful completion when it returns a value of zero).
- | The shell runs the next command only if the current command does not complete.

The syntax for the two ampersand (&&) operator follows:

```
cmd1 && cmd2 && cmd3 && cmd4 && cmd5
```

If *cmd1* succeeds, the shell runs *cmd2*. If *cmd2* succeeds, the shell runs *cmd3*, and on through the series until a command fails or the last command ends. (If any command fails, the shell stops executing the command line).

The syntax for the pipe (|) operator follows:

```
cmd1 | cmd2
```

If *cmd1* fails, then the shell runs *cmd2*. If *cmd1* succeeds, the shell stops executing the command line.

For example, suppose that the command `mysort` is a sorting program that creates a temporary file (`mysort.tmp`) in the course of its sorting process. When the sorting program finishes successfully, it cleans up after itself, deleting the temporary file. If, on the other hand, the program fails, it may neglect to clean up. To ensure deletion of `mysort.tmp`, enter the following command line:

```
$ mysort | rm mysort.tmp  
$
```

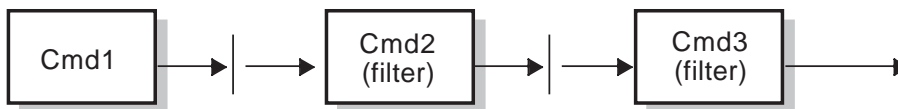
The second command executes only if the first command fails.

7.4.2 Using Pipes and Filters

A **pipe** is a one-way connection between two related commands. One command writes its output to the pipe, and the other process reads its input from the pipe. When two or more commands are connected by the pipe (|) operator, they form a pipeline.

Figure 7-1 represents the flow of input and output through a pipeline. The output of the first command (*cmd1*) is the input for the second command (*cmd2*); the output of the second command is the input for the third command (*cmd3*).

Figure 7-1: Flow Through a Pipeline



ZK-0537U-R

A **filter** is a command that reads its standard input, transforms that input, and then writes the transformed input to standard output. Filters are typically used as intermediate commands in pipelines – that is, they are connected by a pipe (`|`) operator. For example, to cause the `ls` command to list recursively the contents of all directories from the current directory to the bottom of the hierarchy, and then to display the results, enter the following command:

```
$ ls -R | pg
```

In this example, the `pg` command is the filter because it transforms the output from the `ls -R` command and displays it one screen at a time.

Certain commands that are not filters have a flag that causes them to act like filters. For example, the `diff` (compare files) command ordinarily compares two files and writes their differences to standard output. The usual format for `diff` follows:

```
diff file1 file2
```

However, if you use the dash (`-`) flag in place of one of the file names, `diff` reads standard input and compares it to the named file.

In the following pipeline, `ls` writes the contents of the current directory to standard output. The `diff` command compares the output of `ls` with the contents of a file named `dirfile`, and writes the differences to standard output one page at a time (with the `pg` command):

```
$ ls | diff - dirfile | pg
```

In the following example, another kind of filter program (`grep`) is used:

```
$ ls -l | grep r-x | wc -l
12
$
```

In this example, the following takes place:

- The `ls -l` command lists in long format the contents of the current directory.
- The output of `ls -l` becomes the standard input to `grep r-x`, a filter that searches for the files in its standard input for patterns with permissions of `r-x`, and writes all lines that contain the pattern to its standard output.

- The standard output of `grep r-x` becomes the standard input to `wc -l`, which displays the number of files matching the `grep` criteria in the standard input.

To get the same results without using a pipeline, you would have to do the following:

1. Direct the output of `ls -l /user` to a file. For example:

```
$ ls -l >file1
```

2. Use `file1` as input for `grep r-x` and redirect the output of `grep` to another file. For example:

```
$ grep r-x file1 >file2
```

3. Use the output file of `grep` as input for `wc -l`. For example:

```
$ wc -l file2
```

As the preceding procedure demonstrates, using a pipeline is a much easier way to perform the same operations.

Each command in a pipeline runs as a separate process. Pipelines operate in one direction only (left to right), and all processes in a pipeline can run at the same time. A process pauses when it has no input to read or when the pipe to the next process is full.

7.4.3 Grouping Commands

The shell provides two ways to group commands, as shown in Table 7-3.

Table 7-3: Command Grouping Symbols

Symbols	Action
<i>(commands)</i>	The shell creates a subshell to run the grouped commands as a separate process.
<i>{commands}</i>	The shell runs the grouped commands as a unit. Braces can only be used in the Korn and Bourne shells.

The following sections describe the command grouping symbols of Table 7-3 in greater detail.

7.4.3.1 Using Parentheses ()

In the following command grouping, the shell runs the commands enclosed in parentheses as a separate process:

```
$ (cd reports;ls);ls
```

The shell creates a **subshell** (a separate shell program) that moves to the `reports` directory and lists the files in that directory. After the subshell process is complete, the shell lists the files in the current directory (`ls`).

If this command were written without the parentheses, the original shell would move to the `reports` directory, list the files in that directory, and then list the files in that directory again. There would be no subshell and no separate process for the `cd reports;ls` command.

The shell recognizes the parentheses wherever they occur in the command line. To use parentheses literally (that is, without their command-grouping action), quote them by placing a backslash (`\`) immediately before either the open parenthesis [`(`] or the close parenthesis [`)`], for example, `\(.`

For more information on quoting in the shell, see Section 7.4.4.

7.4.3.2 Using Braces { }

Using braces is valid only in the Bourne and Korn shells.

When commands are grouped in braces, the shell executes them without creating a subshell. In the following example, the shell runs the `date` command, writing its output to the `today.grp` file, and then runs the `who` command, writing its output to `today.grp` :

```
$ { date; who }>today.grp
$
```

If the commands were not grouped together with braces, the shell would write the output of the `date` command to the display and the output of the `who` command to the file.

The shell recognizes braces in pipelines and command lists, but only if the left brace is the first character on a command line.

7.4.4 Quoting

Reserved characters are characters such as the left-angle bracket (`<`), the right-angle bracket (`>`), the pipe (`|`), the ampersand (`&`), the asterisk (`*`), and the question mark (`?`) that have a special meaning to the shell. See Chapter 8 for lists of reserved characters for each operating system shell.

To use a reserved character literally (that is, without its special meaning), quote it with one of the three shell quoting conventions, as shown in Table 7-4.

Table 7-4: Shell Quoting Conventions

Quoting Convention	Action
<code>\</code>	Backslash – Quotes a single character.
<code>' '</code>	Single quotes – Quotes a string of characters (except the single quotation marks themselves).
<code>" "</code>	Double quotes – Quotes a string of characters (except <code>\$</code> , <code>\</code> , and <code>)</code> .

The following sections describe the quoting conventions of Table 7-4 in greater detail.

7.4.4.1 Using the Backslash (\)

To quote a single character, place a backslash (`\`) immediately before that character, as in the following:

```
$ echo \  
?  
$
```

This command displays a single question mark (?) character.

7.4.4.2 Using Single Quotes (' ')

When you enclose a string of characters in single quotes, the shell takes every character in the string (except the `'` itself) literally. Single quotes are useful when you do not want the shell to interpret:

- Reserved characters such the dollar sign (`$`), the grave accent (```), and the backslash (`\`)
- Variable names

The following example shows how single quotes are used when you want to display a variable name without having it being interpreted by the shell:

```
$ echo 'The value of $USER is' $USER  
The value of $USER is amy  
$
```

The `echo` command displays the variable name `$USER` when it appears within single quotes, but interprets the value of `$USER` when it appears outside the single quotes.

For information on variable assignments, see Section 7.7.1.

7.4.4.3 Using Double Quotes (" ")

Double quotes (" ") provide a special form of quoting. Within double quotes, the reserved characters dollar sign (\$), grave accent (`), and backslash (\) keep their special meanings. The shell takes literally all other characters within the double quotes. Double quotes are most frequently used in variable assignments.

The following example shows how double quotes are used when you want to display brackets (normally reserved characters) in a message containing the value of the shell variable:

```
echo "<<Current shell is $SHELL>>"
<<Current shell is /usr/bin/csh>>
$
```

For information on variable assignments, see Section 7.7.1.

7.5 The Shell Environment

Whenever you log in, your default shell defines and maintains a unique working environment for you. Your environment defines such characteristics as your user identity, where you are working on the system, and what commands you are running.

Your working environment is defined by both environment variables and shell variables. Your default login shell uses environment variables and passes them to all processes and subshells that you create. Shell variables are valid only for your current shell and are not passed to subshells.

The following sections discuss the shell environment, how it is configured, and how you can tailor it.

7.5.1 The login Program

Whenever you log in, the `login` program is run. This program actually begins your login session using data stored in the `/etc/passwd` file, which contains one line of information about each system user. The `/etc/passwd` file contains your username, your password (in encrypted form), your home directory, and your default shell. For more information on the `/etc/passwd` file, see Chapter 5.

The `login` program runs after you enter your username at the `login:` prompt. It performs the following functions:

- Displays the `Password:` prompt (if you have a password).
- Verifies the username and password you entered against what is contained in the `/etc/passwd` file.
- Assigns default values to the shell environment.

- Starts running the shell process.
- Runs system login scripts and your personal login scripts. See Section 7.6 for more information.

7.5.2 Environment Variables

Your shell environment defines and maintains a unique working environment for you. Most of the characteristics of your working environment are defined by environment variables.

Environment variables consist of a name and a value. For example, the environment variable for your login directory is named *HOME*, and its value is defined automatically when you log in.

Some environment variables are set by the `login` program, and some can be defined in the login script that is appropriate for your shell. For example, if you use the C shell, environment variables will typically be set in the `.cshrc` login script. For more information on login scripts, see Section 7.6.

Table 7-5 lists selected environment variables that can be used by all operating system shells. Most of the values of these variables are set during the login process, and are then passed to each process that you create during your session.

Table 7-5: Selected Shell Environment Variables

Environment Variable	Description
HOME	Specifies the name of your login directory, the directory that becomes the current directory upon completion of a login. The <code>cd</code> command uses the value of <code>HOME</code> as its default value. The <code>login</code> program sets this variable, and it cannot be changed by the individual user.
LOGNAME	Specifies your login name.
MAIL	Specifies the pathname of the file used by the mail system to detect the arrival of new mail. The <code>login</code> program sets this variable based upon your username.
PATH	Specifies the directories and the directory order that your system uses to search for, find, and execute commands. This variable is set by your login scripts.
SHELL	Specifies your default shell. This variable is set by <code>login</code> using the shell specified in your entry in the <code>/etc/passwd</code> file.

Table 7-5: (continued)

Environment Variable	Description
TERM	Specifies the type of terminal you are using. This variable is usually set by your login script.
TZ	Specifies the current time zone and difference from Greenwich mean time. This variable is set by the system login script.
LANG	Specifies the locale of your system, which is comprised of three parts: language, territory, and character code set. The default value is the C locale, which implies English for language, U.S. for territory, and ASCII for code set. LANG can be set in a login script.
LC_COLLATE	Specifies the collating sequence to use when sorting names and when character ranges occur in patterns. The default value is the ASCII collating sequence. LC_COLLATE can be set in a login script.
LC_CTYPE	Specifies the character classification rules used in the ctype functions for the current locale. The default value is the classification for ASCII characters. LC_CTYPE can be set in a login script.
LC_MESSAGES	Specifies the language used for yes/no prompts. The default value is American English, but your system may specify another language.
LC_MONETARY	Specifies the monetary format for your system. The default value is the American format for monetary figures. LC_MONETARY can be set in a login script.
LC_NUMERIC	Specifies the numeric format for your system. The default value is the American format for numeric quantities. LC_NUMERIC can be set in a login script.
LC_TIME	Specifies the date and time format for your system. The default value is the American format for dates and times. LC_TIME can be set in a login script.

Many of these environment variables can be set during the login process by the appropriate login script (see Section 7.6). However, you may reset them as well as set those for which no default values have been provided. See Section 7.7.1 for more information.

For more information on the LANG, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME variables, refer to Appendix C which explains the variables in the context of other system features that support the languages and customs of different countries.

You may also create your own environment variables. For example, some systems have more than one mail program available to users. Assume that `mail` and `mh` are available on your system and that each has its own pathname. As a result, you could define a variable for the pathname of each mail program.

For more information about environment variables specific to each operating system shell, see Chapter 8. For a complete list of operating system shell environment variables, see the `sh(1)`, `csh(1)`, and `ksh(1)` reference pages.

7.5.3 Shell Variables

Shell variables are valid only for your current shell and are not passed to subshells. Consequently, they can be used only in the shell in which they are defined. In other words, they may be thought of as local variables.

Shell variables can be accessed outside of the current shell by becoming environment variables. For more information about environment variables, see Section 7.7.1.

You may also create your own shell variables. For example, some mail programs use the `pager` variable to define the program that displays mail. Suppose that your mail program is `mailx`. You could define the `PAGER` variable to use the `more` program to display your mail.

For information on how to set shell variables, see Section 7.7.1.

7.6 Login Scripts and Your Environment

A login script is a file that contains commands that set up your user environment. There are two kinds of login scripts:

- System login scripts for all users of a particular shell.

These scripts create a default environment for all users and are maintained by your system administrator. The Bourne and Korn shells use a system login script called `/etc/profile`. The C shell uses a script called `/etc/csh.login`. See Table 7-6 for the pathnames of system login scripts.

When you log in, the commands in this file are executed first.

- Local login scripts in your default login directory.

These scripts allow you to tailor your environment, and you maintain the appropriate file. For example, you could change the default search path or shell prompt. If your default shell (see Section 7.3) is the Bourne or Korn shell, the login script file is called `.profile`. The C shell uses the file

called `.login`. A local login script is executed after the system login script.

If the C shell is your default, your environment can be further tailored with the `.cshrc` file. It executes when you log in (before `.login`) and whenever you spawn a subshell. The `.cshrc` file is the C-shell mechanism that automatically makes variables available to subshells.

On startup, the Korn shell will also execute any file pointed to by the `ENV` environment variable. This variable is typically set in the `.profile` file and is set to another file, usually in the `$HOME` directory. Some users prefer to call this file `.kshrc` or `.envfile`. To use such a file, place a line like this in your `.profile` file:

```
ENV=~/.kshrc
```

Such a file typically contains shell variables, alias definitions, and function definitions. This file will be referred to as `.kshrc` for the remainder of this document.

Creating your own login script is not mandatory as the system login script for your shell provides a basic environment. Your system administrator may have created a local login script that you can modify with a text editor.

When you are new to the system, you may want to use the default environment established for you. However, as you become more familiar with the system, you may want to create or modify your own login script.

Table 7-6 lists the system login and local login scripts for each operating system shell. All scripts for a given shell run whenever you log in to your system. In addition, when you enter `csH` at any shell prompt or execute a C shell script, the `.cshrc` file executes and creates an environment for the C subshell.

Table 7-6: System and Local Login Scripts

Shell	Pathname	System Login Script	Local Login Script
Bourne	<code>/usr/bin/sh</code>	<code>/etc/profile</code>	<code>.profile</code>
Korn	<code>/usr/bin/ksh</code>	<code>/etc/profile</code>	<code>.profile</code> <code>ENV</code>
C	<code>/usr/bin/csh</code>	<code>/etc/csh.login</code>	<code>.login</code> <code>.cshrc</code>

To determine if you have any local login scripts in your home directory, use the `ls -a` command. This command displays all files that begin with a dot

(.) as well as all other entries.

The following customization features are commonly set in the local login scripts:

- Terminal characteristics
- Search path and other environment variables
- Shell variables
- Maximum permissions for new files with `umask` (see Chapter 5)
- Allowing or stopping messages to your workstation
- The `trap` command (Bourne and Korn shells only)
- Command aliases, history variables (C and Korn shells only)
- Displaying system status information and other messages
- Checking for mail
- Checking for news

It is a good idea to check the contents of your system login script so that you can avoid duplication in your local login script. For example, if your system login script checks for news, there is no need to do the same in your local login script.

See Chapter 8 for specific examples of Bourne, Korn, and C login scripts.

7.7 Using Variables

All operating system shells use environment and shell variables to define user environment characteristics. As part of the set-up process, your system administrator has provided default environment and shell variable values in the appropriate login scripts.

For most users, the default environment and shell variable values are sufficient. As you become more familiar with the system, however, you may want to modify some values. For example, you may want to reset the variable that defines your shell prompt so that it is more personalized. Or you may want to set a shell variable that specifies a very long directory pathname so that you can save time keying commands that use the directory (see the examples in Section 7.7.1). Or you may find setting variables useful when writing shell procedures. You will find that you may use variables creatively to enhance your work environment.

Some environment variables may be reset and some are read-only and cannot be reset. That is, these variables can be used, but not modified. For more information on this topic, see the appropriate shell reference page; `sh(1)`, `csh(1)`, or `ksh(1)`.

To reset environment variables as well as define your own shell variables, do one of the following:

- Edit the appropriate local login script if you want these values set for you whenever you log in. For more information, see Section 7.6.
- Set the environment variables on the command line if you want these values set only for the current login session.

At any time, you may reference the value of any variable as well as display its value. You may also clear the value of any variable.

7.7.1 Setting Variables

The following sections describe how to set, reference, display, and clear variable values.

7.7.1.1 Bourne and Korn Shell Variables

In the Bourne and Korn shells, you set variables with an assignment statement. The general format for setting variables is the following:

```
name=value
```

The *name* entry specifies the variable name. The *value* entry specifies the value assigned to the variable. Be sure you do not enter spaces on the command line.

For example, you can create a variable called *place* by assigning it a value of U. S. A. with the following statement:

```
$ place='U. S. A.'  
$
```

From then on, you can use the variable *place* just as you would use its value.

For a more useful example, assume that you are using the Bourne shell and that you temporarily want to personalize your shell prompt. The default Bourne shell prompt is a dollar sign (\$) set by the *PS1* environment variable. To set it to What Shall I Do Next? >, enter the following command:

```
$ PS1='What Shall I Do Next? >'  
What Shall I Do Next? >
```

If you want to make the shell prompt available to subshells, enter the

following command:

```
$ export PS1
```

This What Shall I Do Next? > prompt will be in effect throughout your session. If you want to make the new prompt more permanent, enter the same assignment statement and the `export` command in your `.profile` file. When you export a shell variable, it becomes an environment variable.

As another example, to save keying time you want to define a variable for a long pathname that you use often. To define the variable `reports` for the directory `/usr/sales/shoes/women/retail/reports`, enter the following:

```
$ reports=/usr/sales/shoes/women/retail/reports
```

To reference the variable after setting it, enter a dollar sign (\$) before the variable name. For more information on referencing variables, see Section 7.7.2.

You can now use the variable `reports` in any commands you enter during this session. If you want to make this variable permanent, enter the same assignment statement in your `.profile` file.

7.7.1.2 C Shell Variables

In the C shell, you set environment variables with the `setenv` command. The general format of the `setenv` command is the following:

```
setenv name value
```

The *name* entry specifies the variable name. The *value* entry specifies the value assigned to the variable.

For an example of setting the `PATH` environment variable, see Section 7.8.

You set shell variables with the `set` command. The format of the `set` command is:

```
set name=value
```

The *name* entry specifies the variable name. The *value* entry specifies the value assigned to the variable. If the *value* entry contains more than one part (has spaces), enclose the whole expression in single quotes (').

For example, assume that you want to change your prompt. The default C shell prompt is a percent sign (%). To set it to Ready? >, enter the

following on the command line:

```
% set prompt='Ready? >'
Ready? >
```

The `Ready? >` prompt will be in effect throughout your session. If you execute another shell from the `Ready? >` prompt, you will get the new shell's prompt. To make the new prompt permanent, enter the same command in your `.cshrc` file.

7.7.1.3 Setting Variables in All Shells

To set or reset environment or shell variables in any operating system shell, do one of the following:

- Edit the appropriate local login script if you want these values set for you whenever you log in. For more information about login scripts, see Section 7.6.
- Set them on the command line if you want these values set only for the current login session.

7.7.2 Referencing Variables (Parameter Substitution)

To reference the value of a variable in a command line, enter a dollar sign (\$) before the variable name. The dollar sign (\$) causes the shell you are using to substitute the value of the variable for the variable name. This is known as parameter substitution.

For example, assume that you have previously defined the variable `sales` for the long pathname `/user/reports/Q1/march/sales`, and that you want to use this variable with the `cd` command. To do so, enter the `cd` command with the `sales` variable:

```
$ cd $sales
$
```

Then, enter the `pwd` command to verify that the directory is changed:

```
$ pwd
/user/reports/Q1/march/sales
$
```

In this example, the shell substitutes the actual pathname of the directory `/user/reports/Q1/march/sales` for the variable name `sales`.

7.7.3 Displaying the Values of Variables

You can display the value of any variable currently set in your shell. Variable values can be displayed either singly or as a group.

To display the value of a single variable, use the `echo` command in the following general format:

```
echo $variable
```

The *variable* entry identifies the variable you want to display.

For example, assume that you use the Korn shell and want to display the value of the *SHELL* environment variable. To do so, enter the following command:

```
$ echo $SHELL  
/usr/bin/ksh  
$
```

For the Bourne and Korn shells, to display the value of all currently set variables, use the `set` command without any options. For example, the following example lists the currently set values in the Bourne shell (your output will vary):

```
$ set  
EDITOR=emacs  
HOME=/users/chang  
LOGNAME=chang  
MAIL=/usr/mail/chang  
PATH=:/usr/bin:/usr/bin/X11  
PS1=$  
SHELL=/usr/bin/sh  
TERM=xterm  
$
```

For the C shell, to display the value of all currently set shell variables, use the `set` command without any options. To display the value of all currently set environment variables, use the `setenv` command or the `printenv` command without any options.

7.7.4 Clearing the Values of Variables

You may remove the value of most any current variable. Please note, however, that the following variables cannot be cleared:

- *PATH*
- *PS1* (Bourne and Korn shell)
- *PS2* (Bourne and Korn shell)
- *MAILCHECK* (Bourne and Korn shell)
- *IFS* (Bourne and Korn shell)

For more information on these variables, see the appropriate shell reference pages; `sh(1)`, `csh(1)`, or `ksh(1)`.

In the Bourne and Korn shells, you can clear both environment and shell variables with the `unset` command. The format of the `unset` command is:

unset *name*

The *name* entry specifies the variable name.

In the C shell, you clear environment variables with the `unsetenv` command. The format of the `unsetenv` command is:

unsetenv *name*

The *name* entry specifies the variable name.

You clear shell variables with the `unset` command. The format of the `unset` command is:

unset *name*

The *name* entry specifies the variable name.

For an example, assume that you use the Korn shell and have created a variable called *place* and have assigned it a value of `U. S. A..` To clear the variable, enter the following:

```
$ unset place
$
```

For more detailed information about setting and referencing variables, see the appropriate shell reference pages; `sh(1)`, `cs(1)`, or `ksh(1)`.

7.8 How the Shell Finds Commands

Every time you enter a command, your shell searches through a list of directories to find the command. This list of directories is specified by the `PATH` environment variable.

At many installations, system administrators specify default `PATH` directories for new users. However, more experienced users may need to change these `PATH` directories.

The `PATH` variable contains a list of directories to search, separated by colons (:). The order in which the directories are listed is the search order that the shell uses to search for the commands that you enter.

To determine the value of `PATH`, use the `echo` command. For example,

assume that you are using the C shell and have entered the following:

```
% echo $PATH
/usr/bin:/usr/bin/X11
% _
```

This output from the `echo` command (your output may vary) tells you that the search order of the preceding example is the following:

- The `/usr/bin` directory is searched first.
- The `/usr/bin/X11` directory is searched second.

Typically, `PATH` is set as an environment variable in the appropriate login script. In the Bourne and Korn shells, the `PATH` variable is normally set in the `.profile` script. In the C shell, it is normally set in the `.login` script.

If you want to change the search path, you can assign a new value to the `PATH` variable. For example, assume that you use the Bourne shell and that you have decided to use your own versions of some operating system commands. As a result, you want to add `$HOME/usr/bin/` to the search path. To do so, enter the following on the command line if you want the new `PATH` variable value to be in effect for the current login session:

```
$ export PATH=$HOME/usr/bin:/usr/bin:/usr/bin/X11
$
```

If you want this new `PATH` variable value to be in effect for all future sessions, modify the `PATH` variable in your `.profile` script. When you next log in, the changes you have made in your `.profile` script will take effect.

7.9 Using Logout Scripts

You can create a logout script that automatically runs every time you end your session. Just like login scripts, the `.logout` file must reside in your home directory.

You can use logout scripts for the following purposes:

- To clear your screen
- To display a logout message
- To run long background processes after you log out
- To run a file cleanup routine

To create a logout script, do the following:

1. Create a file called `.logout` in your home directory with a text editor.
2. Place the commands you want in the file. See Section 7.9.2 for ideas.

3. Save the text and exit the editor.
4. Enter the following command to ensure that the `.logout` file has the appropriate executable permissions:

```
$ chmod u+x .logout
$
```

Using a `.logout` file is not mandatory; it is a convenience that may enhance your work environment.

7.9.1 Logout Scripts and the Shell

If you are using the C shell, the `.logout` script executes automatically when you log out.

If you are using the Bourne or the Korn shell and want to use a logout script, you must ensure that a special trap is set in your `.profile` script. A **trap** is a command sequence that looks for a specified signal from a terminal, and then runs a specified command or set of commands.

If the following line is not set in your `.profile` script, you must add it with a text editor:

```
trap $HOME/.logout 0
```

This statement tells your system to run the `.logout` script whenever it receives a zero (0) signal, which occurs automatically when you log out.

7.9.2 A Sample .logout File

The following example `.logout` file does the following:

- Clears the screen
- Displays a logout message that provides the name of your system, your username, and the logout time
- Displays a parting message
- Runs a file cleanup routine in the background after you log out

Note that lines beginning with the number sign (#) are comment lines that describe the commands below them.

```
# Clear the screen
clear
```

```
# Display the name of your system, your username,
# and the time and date that you logged out
echo 'hostname' : 'whoami' logged out on `date`
```

```
# Run the find command in the background. This command
# searches your login directory hierarchy for all
```

```
# temporary files that have not been accessed in
# 7 days, and then deletes them.
find ~ -name '*.tmp' -atime +7 -exec rm {} \; &

# A parting message
echo "Good Day. Come Back Soon"
```

7.10 Using Shell Procedures (Scripts)

In addition to running commands from the command line, the shell can read and run commands contained in a file. Such a file is called a **shell procedure** or **shell script**.

Shell procedures are easy to develop, and using them can help you work more efficiently. For example, you may find shell procedures useful because you can place frequently used commands in one file, and then execute them by entering only the name of the procedure. As a result, they are useful for doing repetitious tasks that would normally require entering many commands on the command line.

Because shell procedures are text files that do not have to be compiled, they are easy to create and to maintain.

Each shell has its own native programming language. The following are some programming language features that apply to all shells:

- Storing values in variables
- Testing for predefined conditions
- Executing commands repeatedly
- Passing arguments to a program

For more information on specific programming features of your shell, see Chapter 8.

7.10.1 Writing and Running Shell Procedures

To write and run a shell procedure, do the following:

1. Create a file of the commands you need to accomplish a task. Create this file as you would any text file: with `vi` or another editing program. The file can contain any system command or shell command (described in the `sh(1)`, `csh(1)`, or `ksh(1)` reference pages).
2. Use the `chmod +x` command to give the file `x` (execute) status. For example, the command `chmod g+x reserve` gives execute status to the file named `reserve` for any user in your group (`g`). See Chapter 5 for information on using the `chmod` command.

3. Run the procedure by entering its name. Enter the pathname if the procedure file is not in your current directory.

The following example shows a simple shell procedure named `lss` that sorts `ls -l` command output by file size:

```
# ! /usr/bin/csh
# lss: sort and list
ls -l | sort -n +4
```

Table 7-7 describes each line in `lss`.

Table 7-7: Description of Example Shell Script

Shell Command	Description
<code>#! /usr/bin/csh</code>	Specifies the shell where the shell procedure should run. ^a
<code>#lss: list and sort</code>	Comment line that describes the purpose of the shell procedure.
<code>ls -l sort -n +4</code>	These are the commands in the shell procedure. This procedure lists the files in a directory (<code>ls -l</code>). Output from the <code>ls -l</code> command is then piped to the <code>sort</code> command (<code> sort -n +4</code>). This command skips over the first four columns of the <code>ls -l</code> output, sorts the fifth column (the file size column) numerically, and writes the lines to the standard output.

Table notes:

- a. See Section 7.10.2 for more information.

To run the `lss` procedure, enter `lss`. Sample system output looks similar to the following:

```
$ lss
-rw-rw-rw- 1 larry system 65 Mar 13 14:46 file3
-rw-rw-rw- 1 larry system 75 Mar 13 14:45 file2
-rw-rw-rw- 1 larry system 101 Mar 13 14:44 file1
$
```

7.10.2 Specifying a Run Shell

At times, you may want to specify the shell where a shell procedure should run. This is because of possible syntactic differences between the shells, but is especially true of differences between the C shell and the other shells.

By default, the operating system assumes that any shell procedure you run should be executed in the same shell as your login shell. For example, if your login shell is the Korn shell, by default your shell procedures will run in that same shell.

The ability to override the default is very useful for shell procedures that many users run because it ensures that the procedure executes in the correct shell, regardless of the user's login shell. To change this default run shell, include the following command as the first line of the shell procedure:

```
#!shell_path
```

The *shell_path* entry specifies the full pathname of shell where you want the procedure to run.

For example, if you want a shell procedure to run under the C shell, the first line of the procedure should be the following:

```
#!/usr/bin/csh
```

Shell Features 8

This chapter functions as a reference source for C, Bourne, and Korn shell features. Unlike other chapters of this guide that present conceptual and/or tutorial information, the purpose of this chapter is to provide very brief reference information about each shell.

To get the most out this chapter, you should already be familiar with the introductory shell overview information in Chapter 7.

After completing this chapter, you should be able to:

- Understand the main differences between operating system shells
- Understand specific features of each operating system shell
- Understand the specifics of local login scripts for each shell

8.1 Comparison of C, Bourne, and Korn Shell Features

Table 8-1 compares C, Bourne, and Korn shell selected features.

Table 8-1: C, Bourne, and Korn Shell Features

Feature	Description	C	Bourne	Korn
Shell programming	A programming language that includes features such as loops, condition statements, and variables.	Yes	Yes	Yes
Signal trapping	Mechanisms for trapping interruptions and other signals sent by the operating system.	Yes	Yes	Yes
Restricted shells	A security feature that provides a controlled shell environment with limited features.	No	Yes	No

Table 8-1: (continued)

Feature	Description	C	Bourne	Korn
Command aliases	A feature that allows you to abbreviate long command lines or to rename commands.	Yes	No	Yes
Command history	A feature that stores commands and allows you to edit and reuse them.	Yes	No	Yes
Filename completion	A feature that allows you to enter a portion of a filename and the system automatically completes it or suggests a list of possible choices.	Yes	No	Yes
Command line editing	A feature that allows you to edit a current or previously entered command line.	Yes	No	Yes
Array	The ability to group data and call it by a name.	Yes	No	Yes
Integer arithmetic	The ability to perform arithmetic functions within the shell.	Yes	No	Yes
Job control	Facilities for monitoring and accessing background processes.	Yes	No	Yes

For detailed information on shell features, see the appropriate shell reference pages; `sh(1)`, `csh(1)`, or `ksh(1)`.

8.2 C Shell Features

This section describes the following C shell features:

- Sample `.cshrc` and `.login` scripts
- Metacharacters
- Command history and aliases
- Built-in variables and commands

8.2.1 Sample `.cshrc` and `.login` Scripts

The `.cshrc` login script sets up your C shell environment by defining variables and operating parameters for the local shell process. The `.login` script defines variables and operating parameters that you want executed at the beginning of your session, and that you want to be valid for all shell processes during the current login session.

When you log in, the operating system executes the `.cshrc` file in your home directory first, and the `.login` file second. The `.login` script is executed only when you log in. However, the `.cshrc` file is executed each time you create a subshell.

In the following `.cshrc` script, shell variables, command aliases, and command history variables are set. Table 8-2 explains each part of the script.

```
# Set shell variables
set noclobber
set ignoreeof
set notify

# Set command aliases
alias h 'history \!* | more'
alias l 'ls -l'
alias c clear

# Set history variables
set history=40
set savehist=40

# Set prompt
set prompt = '\! % '
```

Table 8-2: Example .cshrc Script

Command	Description
Shell Variables	
<code>set noclobber</code>	Stops files from being overwritten. If set, places restrictions on output redirection > to ensure that files are not accidentally destroyed, and that >> redirections refer to existing files.
<code>set ignoreeof</code>	Specifies that you cannot use Ctrl/D to end your login session. Instead, you must use either the <code>exit</code> or the <code>logout</code> commands.
<code>set notify</code>	Informs you when background processes have completed.
Command Aliases	
<code>alias h 'history \!* more'</code>	Defines the contents of the command history buffer through the <code>more</code> command. The <code>\!*</code> string specifies that all the history buffer should be piped.
<code>alias l 'ls -l'</code>	Defines a short name, <code>l</code> , for the <code>ls -l</code> command that lists directory files in the long format.
<code>alias c clear</code>	Defines a short name, <code>c</code> , for the <code>clear</code> command that clears your screen.
History Variables	
<code>history=40</code>	Instructs the shell to store the last 40 commands in the history buffer.
<code>savehist=40</code>	Instructs the shell to store the last 40 commands and use them as the starting history for the next login session.
Prompt Variable	
<code>set prompt = '\! % '</code>	Changes your prompt so that it tells you the command number of the current command.

In the following `.login` script, the permissions for file creation are set, the `PATH` environment variable is set, and the editor and printer are specified. Table 8-3 explains each part of the script.

```
# Set file creation permissions
umask 027

# Set environment variables
setenv PATH=/usr/bin:/usr/local/bin:
set cdpath=...:$HOME
setenv EDITOR emacs
setenv MAILHOST boston
setenv PRINTER sales
```

Table 8-3: Example `.login` Script

Command	Description
File Permissions	
<code>umask 027</code>	Specifies the permissions to be subtracted from the default permissions set by the creating program for all new files created. The <code>umask</code> value is subtracted from 777 (for executable programs) or from 666. For an executable program, a <code>umask</code> value of 027 results in all permissions for the owner, read and execute permissions for members of the same group, and no permissions for all others.
Environment Variables	
<code>setenv PATH /usr/bin:/usr/local/bin:</code>	Specifies the search path. In this case, <code>/usr/bin</code> is searched first, and <code>/usr/local/bin</code> is searched second.
<code>set cdpath=...:\$HOME</code>	<code>cdpath</code> is a variable that sets the search path for the <code>cd</code> command. This variable assignment specifies that the <code>cd</code> command should search for the named directory in the current directory (<code>.</code>) first, in the parent directory (<code>..</code>) second, and the home directory (<code>\$HOME</code>) third.

Table 8-3: (continued)

Command	Description
<code>setenv EDITOR emacs</code>	Specifies the <code>emacs</code> editor as the default editor when running a program that allows you to edit a file. For example, various mail programs allow you to use an editor to compose and edit messages.
<code>setenv MAILHOST boston</code>	Specifies <code>boston</code> as your mail handling system.
<code>setenv PRINTER sales</code>	Specifies the printer <code>sales</code> as your default printer.

8.2.2 Metacharacters

Table 8-4 describes C shell metacharacters (characters that have special meaning to the shell).

Table 8-4: C Shell Metacharacters

Metacharacter	Description
Syntactic	
<code>;</code>	Separates commands that should be executed sequentially.
<code> </code>	Separates commands that are part of a pipeline.
<code>&&</code>	Runs the next command if the current command succeeds.
<code> </code>	Runs the next command if the current command fails.
<code>()</code>	Groups commands to run as a separate process in a subshell.
<code>&</code>	Runs commands in the background.
Filename	
<code>/</code>	Separates the parts of a file's pathname.
<code>?</code>	Matches any single character except a leading dot (<code>.</code>).
<code>*</code>	Matches any sequence of characters except a leading dot (<code>.</code>).
<code>[]</code>	Matches any of the enclosed characters.

Table 8-4: (continued)

Metacharacter	Description
~	Specifies a home directory when used at the beginning of filenames.
Quotation	
'...'	Specifies that any of the enclosed characters should be interpreted literally; that is, without their special meaning to the shell.
"..."	Provides a special form of quoting. Specifies that the \$ (dollar sign), `(grave accent), and \ (backslash) characters keep their special meaning, while all other enclosed characters are interpreted literally; that is, without their special meaning to the shell. Double quotes are useful in making variable assignments.
Input/Output	
<	Redirects input.
>	Redirects output to a specified file.
<<	Redirects input and specifies that the shell should read input up to a specified line.
>>	Redirects output and specifies that the shell should add output to the end of a file.
>&	Redirects both diagnostic and standard output and appends them to a file.
>>&	Redirects both diagnostic and standard output to the end of an existing file.
>!	Redirects output and specifies that if the <i>noclobber</i> variable is set (prevents overwriting of files); it should be ignored so that the file can be overwritten.
Substitution	
\$	Specifies variable substitution.
!	Specifies history substitution.
:	Precedes substitution modifiers.
^	Used in special kinds of history substitution.
`	Specifies command substitution.

8.2.3 Command History

The command history buffer stores the commands you enter and allows you to display them at any time. As a result, you can select a previous command, or parts of previous commands, and then reexecute them. This feature may save you time because it allows you to reuse long commands instead of reentering them.

You may want to enter the following three commands in your `.cshrc` file:

- `set history=n`
Creates a history buffer that stores the command lines you enter. The *n* entry specifies the number of command lines you want to store in the history buffer.
- `set savehist=n`
Saves the command lines you entered during the current login session and makes them available for the next login session. The *n* entry specifies the number of command lines you want to store in the history buffer when you log out.
- `set prompt='[\!] % '`
Causes your C shell prompt to display the number of each command line.

To see the contents of the history buffer, use the `history` command. The displayed output will be similar to the following (your output will vary):

```
[18] % history
 3 set history=15
 4 pwd
 5 cd /usr/sales
 6 ls -l
 7 cp report report5
 8 mv /usr/accounts/new .
 9 cd /usr/accounts/new
10 mkdir june
11 cd june
12 mv /usr/accounts/new/june .
13 ls -l
14 cd /usr/sales/Q1
15 vi earnings
16 cd /usr/chang
17 vi status
18 history
[19] % _
```

To reexecute any command in the command history buffer, use the commands listed in Table 8-5. Each command starts with an exclamation point (!), which tells the C shell that you are using commands in the history buffer.

Table 8-5: Reexecuting History Buffer Commands

Command	Description
<code>!!</code>	Reexecutes the previous command.
<code>!<i>n</i></code>	Reexecutes the command specified by <i>n</i> . For example, using the history buffer shown in the previous display, <code>!5</code> reexecutes the <code>cd /usr/sales</code> command.
<code>!-<i>n</i></code>	Reexecutes a previous command relative to the current command. For example, using the history buffer shown in the previous display, <code>!-2</code> invokes command number 17, <code>vi status</code> .
<code>!<i>string</i></code>	Reexecutes the most recent command that has first characters matching those specified by <i>string</i> . For example, using the history buffer shown in the previous display, <code>!cp</code> invokes command number 7, <code>cp report report5</code> .
<code>!?<i>string</i></code>	Reexecutes the most recent command line that has any characters matching those specified by <i>string</i> . For example, using the history buffer shown in the previous display, <code>!?Q1</code> invokes command number 14, <code>cd /usr/sales/Q1</code> .

The command history buffer also allows you to reuse previous command arguments as well as to modify previous command lines. For information on these features, see the `cs(1)` reference page.

8.2.4 Filename Completion

The C shell allows you to enter a portion of a filename or pathname at the shell prompt, and the shell will automatically match and complete the name. This feature saves you time when you are trying to display long, unique filenames.

For example, assume that you have the file `meetings_sales_status` in your current directory. To display a long listing of the file, enter the following command:

```
% ls -l meetings
```

The system displays the following on the same command line:

```
% ls -l meetings_sales_status
```

You can now execute the command by pressing Return.

For more detailed information on filename completion, see the `cs(1)` reference page.

8.2.5 Aliases

The command aliases feature allows you to abbreviate long command lines or rename commands. You do this by creating *aliases* for long command lines that you frequently use.

For example, assume that you often need to move to the directory `/usr/chang/reports/status`. You can create an alias `status`, which will move you to that directory whenever you enter it on the command line.

In addition, aliases allow you to make up more descriptive names for commands. For example, you could define an alias named `rename` for the `mv` command.

To create aliases, use the `alias` command. The format of the `alias` command is:

```
alias aliasname command
```

The *aliasname* entry specifies the name you want to use. The *command* entry specifies either the original command or a series of commands. If the *command* has more than one part (has spaces), enclose the whole expression in single quotes (`' '`).

For example, to create the alias `status` that moves you to the directory `/usr/chang/reports/status`, enter the following command:

```
% alias status 'cd /usr/chang/reports/status'
```

The usual way to define aliases is to make them a permanent part of your environment by including them in your `.cshrc` file. As a result, you can use the aliases whenever you log in or start a new shell. See Section 8.2.1 for an example.

To display all alias definitions, enter the following command:

```
% alias
```

To display the definition of a particular alias, enter the following command:

```
% alias aliasname
```

The *aliasname* entry specifies the particular alias for which you are requesting a definition.

To remove an alias for the current login session, use the `unalias` command. The general format of the `unalias` command is the following:

```
unalias aliasname
```

The *aliasname* entry specifies the alias you want to remove.

To remove an alias for the current and all future login sessions, do the following:

1. Enter the following command:

```
% unalias aliasname
```

The *aliasname* entry specifies the alias you want to remove.

2. Edit the `.cshrc` file and remove the alias definition. Then, save the file.
3. Enter the following command to reexecute the `.cshrc` file:

```
% source .cshrc
```

For complete information on using aliases with the C shell, see the `csh(1)` reference page.

8.2.6 Built-In Variables

The C shell provides variables that can be assigned values. These variables can be very useful for storing values that can be later used in commands. In addition, you can directly affect shell behavior by setting those variables to which the shell itself refers.

Table 8-6 describes selected C shell built-in variables that are of the most interest to general users. For a complete list of C shell built-in variables, see the `csh(1)` reference page.

Table 8-6: Built-In C Shell Variables

Variable	Description
<i>argv</i>	Contains a value or values that can be used by the shell or shell scripts.
<i>cwd</i>	Contains the pathname to your current directory. The value of this variable changes every time you use the <code>cd</code> command.
<i>home</i>	Contains the pathname of your home directory. The default value for this variable is specified in the <code>/etc/passwd</code> file.
<i>ignoreeof</i>	Specifies whether Ctrl/D can be used to log out from the system. If set, you must use either <code>logout</code> or <code>exit</code> to log out. If unset, you may use Ctrl/D to log out. This variable is usually set in the <code>.cshrc</code> file.
<i>cdpath</i>	Specifies alternative directories to be searched by the system when locating subdirectories with the <code>cd</code> , <code>chdir</code> , or <code>pushd</code> commands. This variable is usually set in the <code>.login</code> file.
<i>noclobber</i>	Specifies whether a file can be overwritten. If set, places restrictions on output redirection <code>></code> to ensure that files are not accidentally destroyed, and that <code>>></code> redirections refer to existing files. If set, a file cannot be overwritten. This variable is usually set in the <code>.cshrc</code> file.
<i>notify</i>	Specifies whether you want to be notified when a background process has completed. If set, you are notified; if unset, you are not notified. This variable is usually set in the <code>.cshrc</code> file.
<i>path</i>	Specifies the search path that the shell uses to find commands. This variable is usually set in the <code>.login</code> file.
<i>prompt</i>	Can be used to customize your C shell prompt. This variable is usually set in the <code>.cshrc</code> file.
<i>shell</i>	Specifies the shell to create when a program creates a subshell. This variable is usually set in the <code>.login</code> file.
<i>status</i>	Specifies whether the most recently executed command completed without error (a value of zero is returned) or with an error (a nonzero value is returned).

8.2.7 Built-In Commands

Table 8-7 describes selected C shell commands that are of the most interest to general users. For a complete list of C shell built-in commands, see the `cs(1)` reference page.

Table 8-7: Built-In C Shell Commands

Command	Description
<code>alias</code>	Assigns and displays alias definitions. ^a
<code>bg</code>	Puts a suspended process in the background. ^b
<code>echo</code>	Writes arguments to the shell's standard output. For more information and the command format, see the <code>cs(1)</code> reference page.
<code>fg</code>	Puts a currently running background process in the foreground. ^b
<code>history</code>	Displays the contents of the command history buffer. ^c
<code>jobs</code>	Displays the job number and the PID number of current background processes. ^b
<code>logout</code>	Terminates the login session.
<code>rehash</code>	Tells the shell to recompute the hash table of command locations. Use this command if you add a command to a directory in the shell's search path and want the shell to be able to find it. If you do not use <code>rehash</code> , the command cannot be executed because it was not in the directory when the hash table was originally created.
<code>repeat</code>	Repeats a command a specified number of times. For more information and the command format, see the <code>cs(1)</code> reference page.
<code>set</code>	Assigns and displays shell variable values. ^d
<code>setenv</code>	Assigns environment variable values. ^d
<code>source</code>	Executes commands in a file. This can be used to update the current shell environment. ^e
<code>time</code>	Displays the execution time of a specified command. For more information, see the <code>cs(1)</code> reference page.
<code>unalias</code>	Removes alias definitions. ^a
<code>unset</code>	Removes values that have been assigned to variables. ^d
<code>unsetenv</code>	Removes values that have been assigned to environment variables. ^d

Table notes:

- a. For more information about the `alias` and `unalias` commands, see Section 8.2.5.
- b. For more information about the `bg`, `fg`, and `jobs` commands, see Chapter 6.

- c. For more information about the `history` command, see Section 8.2.3.
- d. For more information about the `set`, `setenv`, `unset`, and `unsetenv` commands, see Chapter 7.
- e. For more information about the `source` command, see Section 8.2.5, and the `cs(1)` reference page.

8.3 Bourne Shell Features

This section describes the following Bourne shell features:

- A sample `.profile` login script
- Metacharacters
- Built-in variables and commands

8.3.1 Sample `.profile` Login Script

If your login shell is the Bourne shell, the operating system executes the `.profile` login script to set up your environment.

The `.profile` login script variables that are exported are passed to any subshells and subprocesses that are created. Variables that are not exported are used only by the login shell.

In the following `.profile` login script, shell variables are set and exported, a trap is set for the logout script, and the system is instructed to display information. Table 8-8 explains each part of the script.

```
# Set PATH
PATH=/usr/bin:/usr/local/bin:
# Export global variables
export PATH
# Set shell variables
PS1='$LOGNAME $ '
CDPATH=.:~:$HOME
# Set up for logout script
trap "echo logout; $HOME/.logout" 0
# Display status information
date
echo "Currently logged in users:" ; users
```

Table 8-8: Example Bourne Shell .profile Script

Command	Description
Set Search Path	
<i>PATH=/usr/bin:/usr/local/bin:</i>	Specifies the search path. In this case, /usr/bin is searched first and /usr/local/bin searched second.
Export Search Path	
<i>export PATH</i>	Specifies that the search path is to be passed to all commands that you execute.
Set Shell Variables	
<i>PS1='\$LOGNAME \$ '</i>	<i>PS1</i> is the variable that specifies the Bourne shell prompt, and its default value is \$. However, this variable assignment specifies that your prompt should be changed to the following: <i>username \$</i> . For example, if your username were <i>amy</i> , your prompt would be the following: <i>amy \$</i> .
<i>CDPATH=.:.:\$HOME</i>	<i>CDPATH</i> is a variable that sets the search path for the <i>cd</i> command. This variable assignment specifies that the <i>cd</i> command should search for the named directory in the current directory (.) first, in the parent directory (..) second, and the home directory (<i>\$HOME</i>) third.
Set Up Logout Script	
<i>trap "echo logout; \$HOME/.logout" 0</i>	Specifies that your shell should display <i>logout</i> and execute your <i>.logout</i> script when the <i>trap</i> command captures the exit signal (0). ^a

Table 8-8: (continued)

Command	Description
Display Status Information	
<i>date</i>	Displays the date and time.

Table notes:

- a. For more information about the `trap` command, see Section 7.9.1.

8.3.2 Metacharacters

Table 8-9 describes Bourne shell metacharacters (characters that have special meaning to the shell).

Table 8-9: Bourne Shell Metacharacters

Metacharacter	Description
Syntactic	
	Separates commands that are part of a pipeline.
&&	Runs the next command if current command succeeds.
	Runs the next command if the current command fails.
;	Separates commands that should be executed sequentially.
::	Separates elements of a case construct.
&	Runs commands in the background.
()	Groups commands to run as a separate process in a subshell.
Filename	
/	Separates the parts of a file's pathname.
?	Matches any single character except a leading dot (.).

Table 8-9: (continued)

Metacharacter	Description
*	Matches any sequence of characters except a leading dot (.).
[]	Matches any of the enclosed characters.
Quotation	
\	Specifies that the following character should be interpreted literally; that is, without its special meaning to the shell.
'...'	Specifies that any of the enclosed characters (except for the ' quote character) should be interpreted literally; that is, without their special meaning to the shell.
"..."	Provides a special form of quoting. Specifies that the \$ (dollar sign), `(grave accent), and \ (backslash) characters keep their special meaning, while all other enclosed characters are interpreted literally; that is, without their special meaning to the shell. Double quotes are useful in making variable assignments.
Input/Output	
<	Redirects input.
>	Redirects output to a specified file.
<<	Redirects input and specifies that the shell should read input up to a specified line.
>>	Redirects output and specifies that the shell should add output to the end of a file.
2>	Redirects diagnostic output to a specified file.
Substitution	
\${...}	Specifies variable substitution.
'...'	Specifies command output substitution.

8.3.3 Built-In Variables

The Bourne shell provides variables that can be assigned values. The shell sets some of these variables, and you can set or reset all of them.

Table 8-10 describes selected Bourne shell built-in variables that are of most interest to general users. For complete information on all Bourne Shell built-in variables, see the `sh(1)` reference page.

Table 8-10: Built-In Bourne Shell Variables

Variable	Description
<i>HOME</i>	Specifies the name of your login directory, the directory that becomes the current directory upon completion of a login. The <code>cd</code> command uses the value of <i>HOME</i> as its default value. <i>HOME</i> is set by the <code>login</code> command.
<i>PATH</i>	Specifies the directories through which your system should search to find and execute commands. The shell searches these directories in the order specified here. Usually, <i>PATH</i> is set in the <code>.profile</code> file.
<i>CDPATH</i>	Specifies the directories that the <code>cd</code> command will search to find the specified argument to <code>cd</code> . If <code>cd</code> 's argument is null, or if it begins with a slash (/), dot (.), or dot dot (..), then <i>CDPATH</i> is ignored. Usually, <i>CDPATH</i> is set in your <code>.profile</code> file.
<i>MAIL</i>	The pathname of the file where your mail is deposited. You must set <i>MAIL</i> , and this is usually done in your <code>.profile</code> file.
<i>MAILCHECK</i>	Specifies in seconds how often the shell checks for mail (600 seconds is the default). If the value of this variable is set to 0, the shell checks for mail before displaying each prompt. <i>MAILCHECK</i> is usually set in your <code>.profile</code> file.
<i>SHELL</i>	Specifies your default shell. This variable should be set and exported by your <code>.profile</code> file.
<i>PS1</i>	Specifies the default Bourne shell prompt, and its default value is <code>\$</code> . <i>PS1</i> is usually set in your <code>.profile</code> file. If <i>PS1</i> is not set, the shell uses the standard primary prompt string.
<i>PS2</i>	Specifies the secondary prompt string – the string that the shell displays when it requires more input after you enter a command line. The standard secondary prompt string is a <code>></code> symbol followed by a space. <i>PS2</i> is usually set in your <code>.profile</code> file. If <i>PS2</i> is not set, the shell uses the standard secondary prompt string.

8.3.4 Built-In Commands

Table 8-11 describes selected Bourne shell commands that are of the most interest to general users. For a complete list of Bourne shell built-in commands, see the `sh(1)` reference page.

Table 8-11: Built-In Bourne Shell Commands

Command	Description
<code>cd</code>	Allows you to change directories. If no directory is specified, the value of the <i>HOME</i> shell variable is used. The <i>CDPATH</i> shell variable defines the search path for this command. ^a
<code>echo</code>	Writes arguments to the standard output. ^b
<code>export</code>	Marks the specified variable for automatic export to the environments of subsequently executed commands.
<code>pwd</code>	Displays the current directory. ^c
<code>set</code>	Assigns and displays variable values. ^d
<code>times</code>	Displays the accumulated user and system times for processes run from the shell.
<code>trap</code>	Runs a specified command when the shell receives a specified signal. ^d
<code>umask</code>	Specifies the permissions to be subtracted for all new files created. ^e
<code>unset</code>	Removes values that have been assigned to variables. ^d

Table notes:

- a. For more information about the `cd` command, see Chapter 4, and the `sh(1)` reference page.
- b. For more information about the `echo` command, see Section 8.3.1 and the `sh(1)` reference page.
- c. For more information about the `pwd` command, see Chapter 2.
- d. For more information about the `set`, `trap`, and `unset` commands, see Chapter 7.
- e. For more information about the `umask` command, see Chapter 5 and Section 8.2.1.

8.4 Korn Shell Features

This section describes the following Korn shell features:

- Sample `.profile` and `.kshrc` login scripts
- Metacharacters
- Command history

- Editing command lines
- Filename completion
- Aliases
- Built-in variables and commands

8.4.1 Sample .profile and .kshrc Login Scripts

If your login shell is the Korn shell, the operating system processes the `.profile` login script in your home directory. The `.profile` login script defines environment variables. These variables are used by your login shell as well as any subshells and subprocess that are created. The `.profile` login script is executed only when you log in.

The `.kshrc` login script sets up your Korn shell environment by defining variables and operating parameters for the local shell process. It is executed each time you create a subshell.

Note

Before creating a `.kshrc` file in your home directory, make sure that the `ENV=$HOME/.kshrc` environment variable is set and exported in your `.profile`. Once this is done, the `.kshrc` login script will execute each time you log in and each time you create a subshell.

In the following `.profile` login script, global environment variables are set and exported, and shell variables are set. Table 8-12 explains each part of the script.

```
# Set environment variables
PATH=/usr/bin:/usr/local/bin:
ENV=$HOME/.kshrc
EDITOR=vi
FCEDIT=vi
PS1="'hostname' [!] $ "

# Export global variables
export PATH ENV EDITOR FCEDIT PS1

# Set mail variables
MAIL=/usr/spool/mail/$LOGNAME
MAILCHECK=300
```


Table 8-12: Example Korn Shell .profile Script

Command	Description
Set Environment Variables	
<code>PATH=/usr/bin:/usr/local/bin</code>	Specifies the search path. In this case, <code>/usr/bin</code> is searched first and <code>/usr/local/bin</code> searched second.
<code>ENV=\$HOME/.kshrc</code>	Specifies <code>\$HOME/.kshrc</code> as the login script.
<code>EDITOR=vi</code>	Specifies <code>vi</code> as the default editor for command line editing at the shell prompt and for filename completion.
<code>FCEDIT=vi</code>	Specifies <code>vi</code> as the default editor for the <code>fc</code> command. ^a
<code>PS1="`hostname` [!] \$ "</code>	<i>PS1</i> is the variable that specifies the Korn shell prompt, and its default value is <code>\$</code> . However, this variable assignment specifies that your prompt should be changed to the following: the output of the <code>hostname</code> command, followed by the command number of the current command, followed by the dollar sign (<code>\$</code>). For example, if the name of your system is <code>boston</code> , and the current command is numbered <code>30</code> , your prompt would be the following: <code>boston[30] \$</code> .
Export Global Variables	
<code>export PATH ENV EDITOR FCEDIT PS1</code>	Specifies that the values of the <i>PATH</i> , <i>ENV</i> , <i>EDITOR</i> , <i>FCEDIT</i> , and <i>PS1</i> variables should be exported to all subshells.

Table 8-12: (continued)

Command	Description
Set Mail Variables	
<code>MAIL=/usr/spool/mail/\$LOGNAME</code>	Specifies the pathname of the file used by the mail system to detect the arrival of new mail. In this case, the mail system would look in your username subdirectory under the <code>/usr/spool/mail</code> directory.
<code>MAILCHECK=300</code>	Specifies that the shell should check for mail every 300 seconds (5 minutes).

Table notes:

- a. For information on the `fc` command, see Section 8.4.4.

In the following `.kshrc` login script, shell variables, command aliases, and command history variables are set, as well as the permissions for file creation. Table 8-13 explains each part of the script.

```
# Set shell variables
set -o monitor
set -o trackall

# Set command aliases
alias rm='rm -i '
alias rename='mv '
alias h 'history \!* | more'
alias l 'ls -l'
alias c clear

# Set history variables
HISTSIZE=40

# Set file creation permissions
umask 027
```

Table 8-13: Example .kshrc Script

Command	Description
Shell Variables	
<code>set -o monitor</code>	Specifies that the shell should monitor all background processes and display a completion message when the process finishes.
<code>set -o trackall</code>	Specifies that the shell should track all commands that you execute. Once a command is tracked, the shell stores the location of the command and finds the command more quickly the next time you enter it.
Command Aliases	
<code>alias rm='rm -i'</code>	Specifies the use of the <code>-i</code> option (which prompts you for file deletion) with the <code>rm</code> command.
<code>alias rename='mv'</code>	Specifies <code>rename</code> as a new name for the <code>mv</code> command.
<code>alias h='history' \!* more'</code>	Defines a command that pipes the contents of the command history buffer through the <code>more</code> command. The <code>\!*</code> string specifies that all of the history buffer should be piped.
<code>alias l='ls -l'</code>	Defines a short name for the <code>ls -l</code> command that lists directory files in the long format.
<code>alias c='clear'</code>	Defines a short name for the <code>clear</code> command that clears your screen.
History Variables	
<code>HISTSIZE=40</code>	Instructs the shell to store the last 40 commands in the history buffer.

Table 8-13: (continued)

Command	Description
Set File Creation Permissions	
<code>umask 027</code>	Specifies the maximum permissions for all new files created. This command provides all permissions for the owner, read, and execute permissions for members of the same group, and no permissions for all others. The umask is not inherited by subshells.

8.4.2 Metacharacters

Table 8-14 describes Korn shell metacharacters (characters that have special meaning to the shell).

Table 8-14: Korn Shell Metacharacters

Metacharacter	Description
Syntactic	
	Separates commands that are part of a pipeline.
&&	Runs the next command if the current command succeeds.
	Runs the next command if the current command fails.
;	Separates commands that should be executed sequentially.
::	Separates elements of a case construct.
&	Runs commands in the background.
()	Groups commands in a subshell as a separate process.
{ }	Groups commands without creating a subshell.
Filename	
/	Separates the parts of a file's pathname.
?	Matches any single character except a leading dot (.).
*	Matches any character sequence except a leading dot (.).
[]	Matches any of the enclosed characters.
~	Specifies a home directory when it begins a filename.

Table 8-14: (continued)

Metacharacter	Description
Quotation	
\	Specifies that the following character should be interpreted literally; that is, without its special meaning to the shell.
'...'	Specifies that any of the enclosed characters (except for the ') should be interpreted literally; that is, without their special meaning to the shell.
"..."	Provides a special form of quoting. Specifies that the dollar sign (\$), ` (grave accent), \ (backslash), and) (close parenthesis) characters keep their special meaning, while all other enclosed characters are interpreted literally; that is, without their special meaning to the shell. Double quotes (" ") are useful in making variable assignments.
Input/Output	
<	Redirects input.
>	Redirects output to a specified file.
<<	Redirects input and specifies that the shell should read input up to a specified line.
>>	Redirects output and specifies that the shell should add output to the end of a file.
>&	Redirects both diagnostic and standard output and appends them to a file.
Substitution	
\${...}	Specifies variable substitution.
%	Specifies job number substitution.
'...'	Specifies command output substitution.

8.4.3 Command History

The command history buffer stores the commands you enter and allows you to display them at any time. As a result, you can select a previous command, or parts of previous commands, and then reexecute them. This feature may save you time because it allows you to reuse long commands instead of reentering them.

To see the contents of the history buffer, use the `history` command. The displayed output will be similar to the following (your output will vary):

```
[18] $ history
 3 ls -l
 4 pwd
 5 cd /usr/sales
 6 ls -l
 7 cp report report5
 8 mv /usr/accounts/new .
 9 cd /usr/accounts/new
10 mkdir june
11 cd june
12 mv /usr/accounts/new/june .
13 ls -l
14 cd /usr/sales/Q1
15 vi earnings
16 cd /usr/chang
17 vi status
[19] $
```

To reexecute any command in the command history buffer, use the commands listed in Table 8-15. Each command starts with the letter `r`.

Table 8-15: Reexecuting History Buffer Commands

Command	Description
<code>r</code>	Reexecutes the previous command
<code>r n</code>	Reexecutes the command specified by <i>n</i> . For example, using the history buffer shown in the previous display, <code>r 5</code> reexecutes the <code>cd /usr/sales</code> command.
<code>r -n</code>	Reexecutes a previous command relative to the current command. For example, using the history buffer shown in the previous display, <code>r-2</code> invokes command number 16, <code>cd /usr/chang</code> .
<code>r string</code>	Reexecutes the most recent command that has first characters matching those specified by <i>string</i> . For example, using the history buffer shown in the previous display, <code>r cp</code> invokes command number 7, <code>cp report report5</code> .

For more information on reexecuting history buffer commands, see the `ksh(1)` reference page.

If you want to increase or decrease the number of commands stored in your history buffer, set the `HISTSIZE` variable in your `.profile` file. This variable has the following format:

HISTSIZE=*n*

The *n* entry specifies the number of command lines you want to store in the history buffer.

For example, to store 15 commands in the history buffer, use the following command:

```
HISTSIZE=15
```

The Korn shell also allows you to edit current command lines as well as reuse those already entered in the command history buffer. To use this feature, you must know how to use a text editor such as `vi` or `emacs`. For information on these features, see the following section.

8.4.4 Command Line Editing Using the `fc` Command

The Korn shell allows you to list and/or edit the command lines in your command history buffer. As a result, you may modify any element of a previous command line and then reexecute the command line.

The command line editing functions for the Korn shell are extensive. This section covers only the most basic functions. For more detailed information, see the `ksh(1)` reference page.

To display the command history buffer and/or to edit its contents, use the built-in command `fc` (fix command). The `fc` command has the following two formats:

1.

```
fc [ -e editor ] [ -nlr ] [ first ] [ last ]
```

This command format allows you to display and/or edit any number of command lines in your buffer.

- The `-e editor` entry specifies the editor (usually `vi` or `emacs`) you want to use in editing the command line. If you do not specify `-e`, the `fc` command displays the lines, but does not allow you to edit them.
- The `-n` flag specifies that you want to list the command lines in the buffer without numbers. The `-l` flag specifies that you want to list the command lines in the buffer with numbers. If you do not specify a line number or a range of line numbers, the last 16 lines you entered will be listed.
- The `-r` flag specifies that you want to list the command in the buffer in reverse order.
- The `first` and `last` entries specify a range of command lines in the buffer. You may specify them either with numbers or with strings.

If you want to specify a default editor for the `-e` flag, define the `FCEDIT` variable in your `.profile` script. For example, if you want to make `emacs` your default editor, enter the following variable definition:

```
FCEDIT=emacs
```

2.

```
fc -e - [ old=new ] [ string ]
```

This command allows you to immediately replace an `old` string with a `new` string within any previous command line.

- The `-e` entry specifies that you want to make a replacement.
- The `old=new` entry specifies that you want to replace the `old` string with the `new` string.
- The `string` entry specifies that the Korn shell should make the edit to the most recent command line in the buffer containing the `string`.

The following section contains some examples of `fc` use.

The Korn shell also allows you to edit individual command lines at the shell prompt by using a command set similar to the `vi` or the `emacs` editors. For more information on this feature, see the `ksh(1)` reference page.

8.4.4.1 Examples of Command Line Editing

Example 1: Displaying Command Lines in the Command History Buffer

To display command lines 15 to 18, enter the following command:

```
$ fc -l 15 18
15 ls -la
16 pwd
17 cd /u/ben/reports
18 more sales
$
```

You may also list the same command lines by specifying command strings instead of line numbers, as in the following example:

```
$ fc -l ls more
15 ls -la
16 pwd
17 cd /u/ben/reports
18 more sales
$
```


Example 2: Editing and Executing Command Lines

To display and edit command lines 15 to 18 with the `vi` editor, enter the following command:

```
$ fc -e vi 15 18
ls -la
pwd
cd /u/ben/reports
more sales
~
~
~
~
```

After making your edits, write and exit the file with the `:wq!` command. The command lines in the file are then reexecuted.

Example 3: Replacing and Reexecuting Command Lines

Assume that you have just entered the `echo hello` command, and now want to replace `hello` with `goodbye`. To do the replacement and reexecute the command line, enter the following command:

```
$ echo hello
hello
$ fc -e - hello=goodbye echo
echo goodbye
goodbye
```

For more detailed information on the `fc` command and command line editing, see the `ksh(1)` reference page.

8.4.5 Filename Completion

The Korn shell allows you to enter a portion of a filename or pathname at the shell prompt and the shell will automatically match and complete the name. If there is more than one filename or pathname that matches the criterion, the shell will provide you with a list of possible matches.

To activate the filename completion mechanism, define the `EDITOR` variable in your `.profile` file. For example, if you want to use the `vi` editor, enter the following variable definition in your `.profile` file:

```
EDITOR=vi
```

To demonstrate how filename completion works, assume that your editor is `vi` and that you have the `salesreport1`, `salesreport2`, and `salesreport3` files in your current directory. To display a long listing

and to activate filename completion, enter the following command:

```
$ ls -l salesreport Escape =
1) salesreportfeb
2) salesreportjan
3) salesreportmar
$ ls -l salesreport
```

The system redisplay your command, and the cursor is now at the end of `salesreport`. If you want to choose `salesreportjan`, type `a` (the `vi` append command) followed by `jan`, then press Return. The listing for `salesreportjan` will be displayed.

For more detailed information on filename completion, see the `ksh(1)` reference page.

8.4.6 Aliases

The command aliases feature allows you to abbreviate long command lines or rename commands. You do this by creating aliases for long command lines that you frequently use.

For example, assume that you often need to move to the directory `/usr/chang/reports/status`. You can create an alias `status`, which will move you to that directory whenever you enter it on the command line.

In addition, aliases allow you to make up more descriptive names for commands. For example, you could define an alias named `rename` for the `mv` command.

To create aliases, use the `alias` command. The general format of the `alias` command is the following:

```
alias aliasname=command
```

The *aliasname* entry specifies the name you want to use. The *command* entry specifies either the original command or a series of commands. If the *command* has more than one part (has spaces), enclose the whole expression in single quotes.

For example, to create the alias `status` that moves you to the directory `/usr/chang/reports/status`, enter the following command:

```
alias status='cd /usr/chang/reports/status'
```

The usual way to define aliases is to place them in your `.kshrc` file so that you can use them whenever you log in or start a new shell. See Section 8.4.1 for an example.

To display all alias definitions, enter the following command:

```
$ alias
```

To display the definition of a particular alias, enter the following command:

```
$ alias aliasname
```

The *aliasname* entry specifies the particular alias for which you are requesting a definition.

The Korn shell allows you to export the aliases you create. Variables that are exported are passed to any subshells that are created so that when you execute a shell procedure or new shell, the alias remains defined. (Variables that are not exported are used only by the login shell.)

To export an alias, use the following form of the `alias` command:

```
alias -x aliasname=command
```

The `-x` flag specifies that you want to export the alias. The *aliasname* entry specifies the name you want to use. The *command* entry specifies either the original command or a series of commands. If the *command* has more than one part, enclose the whole expression in single quotes.

For example, to export an alias definition for the `rm` command, enter the following:

```
alias -x rm='rm -i '
```

You can enter the preceding command in one of two ways:

- Edit the `.kshrc` or `.profile` file if you want an alias exported whenever you log in.
- Export an alias on the command line if you want the alias exported only for the current login session.

To remove an alias for the current login session, use the `unalias` command. The general format of the `unalias` command is the following:

```
unalias aliasname
```

The *aliasname* entry specifies the alias you want to remove.

To remove an alias for the current and all future login sessions, do the following:

1. Enter the following command:

```
$ unalias aliasname
```

The *aliasname* entry specifies the alias you want to remove.

2. Edit the `.kshrc` file (or the file on your system that contains alias definitions) and remove the alias definition. Then, save the file.

3. Enter the following command to reexecute the `.kshrc` file:

```
$ source .kshrc
```

The Korn shell provides additional aliasing features. For complete information on using aliases with the Korn shell, see the `ksh(1)` reference page.

8.4.7 Built-In Variables

The Korn shell provides variables that can be assigned values. The shell sets some of these variables, and you can set or reset all of them.

Table 8-16 describes selected Korn shell built-in variables that are of the most interest to general users. For complete information on all Korn shell built-in variables, see the `ksh(1)` reference page.

Table 8-16: Built-In Korn Shell Variables

Variable	Description
<i>HOME</i>	Specifies the name of your login directory. The <code>cd</code> command uses the value of <i>HOME</i> as its default value. In Korn shell procedures, use <i>HOME</i> to avoid having to use full pathnames – something that is especially helpful if the pathname of your login directory changes. <i>HOME</i> is set by the <code>login</code> command.
<i>PATH</i>	Specifies the directories through which your system should search to find and execute commands. The shell searches these directories in the order specified here. Usually, <i>PATH</i> is set in the <code>.profile</code> file.
<i>CDPATH</i>	Specifies the directories that the <code>cd</code> command will search to find the specified argument to <code>cd</code> . If the <code>cd</code> argument is null, or if it begins with a slash (<code>/</code>), dot (<code>.</code>), or dot dot (<code>..</code>), then <i>CDPATH</i> is ignored. Usually, <i>CDPATH</i> is set in your <code>.profile</code> file.
<i>MAIL</i>	The pathname of the file where your mail is deposited. <i>MAIL</i> is usually set in your <code>.profile</code> file.
<i>MAILCHECK</i>	Specifies in seconds how often the shell checks for mail (600 seconds is the default). If the value of this variable is set to 0, the shell checks for mail before displaying each prompt. <i>MAILCHECK</i> is usually set in your <code>.profile</code> file.
<i>SHELL</i>	Specifies your default shell. This variable should be set and exported by your <code>.profile</code> file.

Table 8-16: (continued)

Variable	Description
<i>PS1</i>	Specifies the default Korn shell prompt, and its default value is the dollar sign (\$). <i>PS1</i> is usually set in your <code>.profile</code> file. If <i>PS1</i> is not set, the shell uses the standard primary prompt string.
<i>PS2</i>	Specifies the secondary prompt string – the string that the shell displays when it requires more input after entering a command line. The standard secondary prompt string is a <code>></code> symbol followed by a space. <i>PS2</i> is usually set in your <code>.profile</code> file. If <i>PS2</i> is not set, the shell uses the standard secondary prompt string.
<i>HISTFILE</i>	Specifies the pathname of the file that will be used to store the command history. This variable is usually set in your <code>.profile</code> file.
<i>EDITOR</i>	Specifies the default editor for command line editing at the shell prompt and for filename completion. This variable is usually set in your <code>.profile</code> file.
<i>FCEDIT</i>	Specifies the default editor for the <code>fc</code> command. This variable is usually set in your <code>.profile</code> file.
<i>HISTSIZE</i>	Specifies the number of previously entered commands that are accessible by this shell. The default is 128. This variable is usually set in your <code>.kshrc</code> file.

8.4.8 Built-In Commands

Table 8-17 describes selected Korn shell commands that are of the most interest to general users. For a complete list of Korn shell built-in commands, see the `ksh(1)` reference page.

Table 8-17: Built-In Korn Shell Commands

Command	Description
<code>alias</code>	Assigns and displays alias definitions. ^a
<code>cd</code>	Allows you to change directories. If no directory is specified, the value of the <code>HOME</code> shell variable is used. The <code>CDPATH</code> shell variable defines the search path for this command. ^b
<code>echo</code>	Writes arguments to the standard output. For more information and the command format, see the <code>ksh(1)</code> reference page.

Table 8-17: (continued)

Command	Description
<code>export</code>	Marks the specified variable for automatic export to the environments of subsequently executed commands. ^c
<code>fc</code>	Allows you to display, edit, and reexecute the contents of the command history buffer. ^d
<code>history</code>	Displays the contents of the command history buffer. ^a
<code>jobs</code>	Displays the job number and the PID number of current background processes.
<code>pwd</code>	Displays the current directory. ^e
<code>set</code>	Assigns and displays variable values. ^f
<code>times</code>	Displays the accumulated user and system times for processes run from the shell.
<code>trap</code>	Runs a specified command when the shell receives a specified signal. ^f
<code>umask</code>	Specifies the permissions to be subtracted from the default permissions set by the creating program for all new files created. ^g
<code>unalias</code>	Removes alias definitions. ^a
<code>unset</code>	Removes values that have been assigned to variables. ^f

Table notes:

- a. For more information about the `alias`, `history`, and `unalias` commands, see Section 8.4.6.
- b. For more information about the `cd` command, see Chapter 4 and the `ksh(1)` reference page.
- c. For more information about the `export` command, see Section 8.4.1 and the `ksh(1)` reference page.
- d. For more information about the `fc` command, see Section 8.4.4.
- e. For more information about the `pwd` command, see Chapter 2.
- f. For more information about the `set`, `trap`, and `unset` commands, see Chapter 7.
- g. For more information about the `umask` command, see Chapter 5 and Section 8.4.1.

Using the System V Habitat **9**

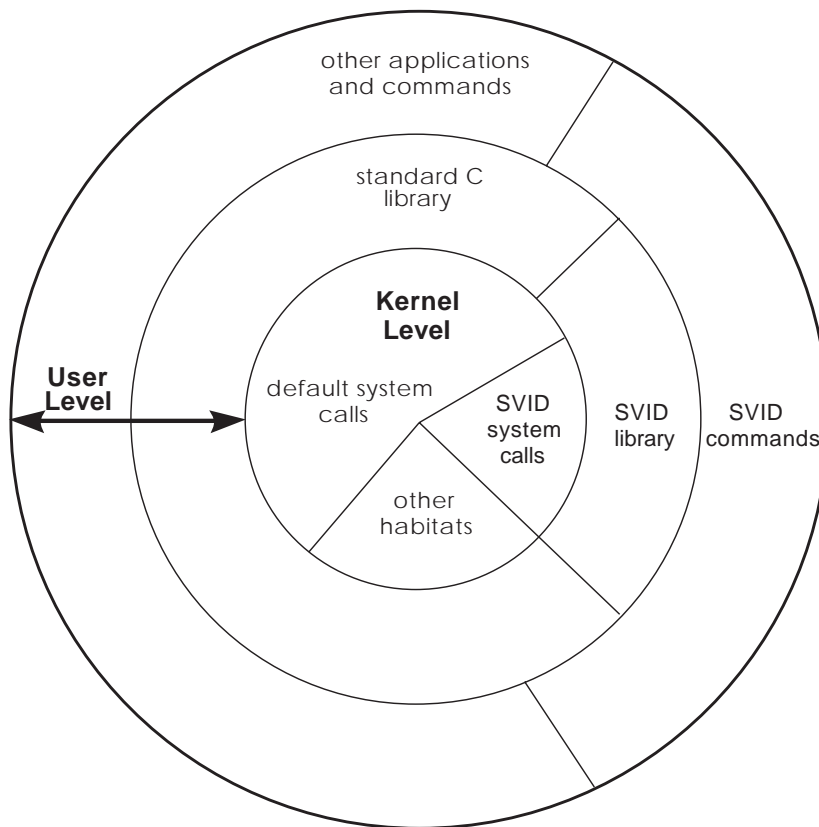
The System V habitat consists of alternate versions of commands, subroutines, and system calls that support the source code interfaces and runtime behavior for all components of the Base System and Kernel Extension as defined in the System V Interface Definition (SVID). This implementation of the System V habitat supports all SVID 2 functions and SVID 3 functions. Note that the System V habitat does not contain alternate versions of default system commands, subroutines, and system calls that already meet the SVID requirement.

Using the System V habitat lets you override the default system commands and functions with corresponding System V commands and functions (system calls and subroutines). You can access the habitat in two ways:

- Specify the absolute paths of the System V commands and libraries.
- Define the `PATH` environment variable to search the System V habitat for commands before it searches the default system locations. To set your `PATH` environment variable, modify your `.profile` or `.login` and `.cshrc` files as described in Section 9.1.

Because the System V system calls are not layered over the system calls in the default system, applications that are built using the system calls in the System V habitat run with virtually no performance overhead. Figure 9-1 illustrates the System V habitat placement within the default operating system and shows that the System V system calls reside at the kernel level.

Figure 9-1: System V Habitat



ZK-0849U-R

The following sections describe how to set up your environment to access the System V habitat and how it works.

9.1 Setting Up Your Environment

To automatically access the System V habitat when you log on, you must add a command line to your `.profile` file if you use the Bourne or Korn shell, or to your `.login` and `.cshrc` files if you use the C shell. The command line modifies the `PATH` environment variable, which causes the System V habitat to be searched before the standard default locations on the system are searched, such as `/bin` or `/usr/bin`. The System V habitat scripts are as follows:

- `/etc/svid2_profile` (for the Bourne or Korn shells)
`/etc/svid2_login` (for the C shell)
Specifies SVID 2 behavior when placed in either your `.profile` or `.login` and `.cshrc` files, respectively.
- `/etc/svid3_login` (for the Bourne or Korn shell)
`/etc/svid3_profile` (for the C shell)
Specifies SVID 3 behavior when placed in either your `.profile` or `.login` and `.cshrc` files, respectively.

For example, if you use the Bourne or Korn shell and you want to specify SVID 2 behavior, edit the `.profile` file and add the following command line:

```
if [ -f /etc/svid2_profile ]
then
    . /etc/svid2_profile
fi
```

If you use the C shell and you want to specify SVID 2 behavior, edit the `.login` and `.cshrc` files and add the following command line:

```
if ( -e /etc/svid2_login ) then
    source /etc/svid2_login
endif
```

The dot (`.`) and `source` commands are shell-specific. See the appropriate reference page for more information.

9.2 How the System V Habitat Access Works

Whether you choose the script that specifies SVID 2 behavior or the script that specifies SVID 3 behavior, both establish the System V habitat as follows:

- Defines `SVID2PATH`, a System V only variable, to be the contents of `/etc/svid2path` or `/etc/svid3path`. The `/etc/svid2path` and `/etc/svid3path` files contain the path definition for SVID 2 and SVID 3, respectively.
- Adds `SVID2PATH/bin` and `SVID2PATH/sbin` to the beginning of your current `PATH`.
- Exports both `SVID2PATH` and `PATH`.
- Sets the `TZ` variable to the appropriate value. If you select SVID 2 behavior, it also sets the `TZC` variable to the appropriate value. See the *System Administration* guide for more information on time zone formats.

Hence, if you need to determine the location of the System V habitat on your system, run the `cat(1)` command on the `/etc/svid2path` or

/etc/svid3path file.

By using the System V habitat scripts to alter the PATH environment variable, the System V habitat path can be changed without an administrator updating each user's .profile or .login and .cshrc files. The administrator simply updates the /etc/svid2path and /etc/svid3path files to enable global definitions.

To further illustrate how the System V habitat script sets a PATH, look at the following .profile file which specifies the System V habitat script for SVID 2:

```
stty erase DEL kill ^U intr ^C quit ^X echo
TERM=vt100
PATH=$HOME/bin:/usr/lib:/bin
MAIL=/usr/mail/$LOGNAME
EDITOR=vi
export MAIL PATH TERM EDITOR
if [ -f /etc/svid2_profile ]
then
    . /etc/svid2_profile
fi
```

In this example, assume that the path of the System V habitat is /usr/opt/s5 as reflected by the contents of /etc/svid2path and that your login directory is /usr/users/xxx. When you display the PATH after logging in with the preceding .profile file, the result would show that the path to the System V habitat has been prepended to the PATH set in the third line of the .profile file as follows:

```
% echo $PATH Return
/usr/opt/s5/bin:/usr/opt/s5/sbin:/usr/users/xxx/bin:/usr/lib:/bin
```

Hence, when you issue a shell command, the System V habitat is searched first. If the command is not found, the specified paths are searched.

9.3 Compatibility for Shell Scripts

Compatibility for your shell scripts is achieved by altering your shell's PATH environment variable (as explained in Section 9.1). Therefore, the System V habitat is searched before the default system locations. If your PATH variable is set for the System V habitat, your scripts are System V compatible regardless of whether you use the Bourne, Korn, or C shell.

9.4 System V Habitat Command Summary

Table 9-1 summarizes the behavior of user commands in the System V habitat that have options or features that differ from the default system versions. For a complete explanation of the commands in the habitat, refer to the reference page for each command.

Table 9-1: User Commands Summary

Command	System V Behavior
chmod(1)	Ignores the <i>umask</i> value when the <i>who</i> string is omitted, behaving as though <i>a</i> is the <i>who</i> value when you use the symbolic form of this command.
df(1)	Accepts the <i>-t</i> option, which prints space totals, and accepts an optional file system name or device name.
ln(1)	Accepts the <i>-f</i> option, which silently removes existing destination pathnames before creating the specified link.
ls(1)	Produces multicolumn output only if the <i>-C</i> option is specified. Also, the <i>-s</i> option causes file sizes to be reported in 512 byte units instead of 1024 byte units.
mailx(1) and Mail(1)	Includes the capabilities of the System V <i>mailx</i> command.
sum(1)	Uses the word-by-word algorithm by default; uses the byte-by-byte algorithm if the <i>-r</i> option is specified. The default use of the checksum algorithms for the System V <i>sum</i> command is the reverse of the default system version of the <i>sum</i> command.
tr(1)	Includes the <i>-A</i> option whenever you specify the <i>-c</i> option. The <i>-A</i> option causes only the characters in the octal range of 1 to 377 to be complemented.

Obtaining Information About Network Users and Hosts 10

This chapter describes how to find information about local and remote users and hosts before you begin communication or file transfer tasks. The commands described in this chapter will enable you to:

- Learn about your own network connection (`who am i` command, Section 10.1)
- Determine who is currently logged in to the local system and from where they are logged in (`who` command, Section 10.1)
- Find additional information about another user, if available; for example, full name, office location, phone number, projects, or plans (`finger` command, Section 10.2.1)
- Determine whether a user can be reached using either the `talk` or `write` commands (`finger` command, Section 10.2.1 and Section 10.2.2)
- Analyze and sort information about remote host usage (`ruptime` command, Section 10.3)
- Determine who is currently logged in to a remote host (`rwho` command, Section 10.4)
- Determine whether a remote host is online (`ping` command, Section 10.5)

Note

The commands described in this chapter are, like all TCP/IP operations, subject to the security features on the local and remote hosts. If they do not work as stated here or in the related reference pages, see your system administrator.

10.1 Identifying Users on the Local Host

When you log in to a host computer by providing a user name and password, you have a unique identity. To verify this information for your own network connection, you can use a version of the `who` command called `who am i` to display the following information about you:

- Login name
- Terminal name (line)
- Time of login
- Computer from which the network connection came

For example, user `lennon` might enter the `who am i` command at the system prompt (`%`) and read the following output:

```
% who am i
lennon      tty0          Jul 15 14:17      (walrus)
```

In this example, user `lennon` logged in from host `walrus` at 2:17 in the afternoon of July 15. The line is `tty0`, and `walrus` is the name for this line, from which the network connection came.

The `who am i` command can help you keep track of the sessions you have running on your workstation. Some sessions may be remote logins to another host by yourself or by someone with whom you are working. See the `who(1)` reference page for more information about the `who am i` command.

To find out if other users are logged in to the same local host, use the `who` command. In the following example, `lennon` enters the `who` command at the prompt of local host `london`, and learns that three other users are currently logged in to `london` from different nodes:

```
london% who
lennon      tty0          Jul 15 08:17      (walrus)
elvis       tty2          Jul 15 07:55      (velvet)
burdon      tty1          Jul 15 09:02      (animal)
sarjan      tty4          Jul 14 16:47      (pepper)
```

The output from the `who` command is the same as that from the `who am i` command.

10.2 Obtaining Information About Network Users

The `finger` command and its options enable you to display information about users with accounts on local or remote hosts. The specified host must be running a `fingerd` daemon server or have the `inetd` daemon configured to start `fingerd`. See your system administrator if the `finger` command does not work as described in the `finger(1)` reference page.

The `finger` command has the following syntax:

```
finger [ [ option... ] [ user... ] [ user@host_name... ] ]
```

If you use the `finger` command without specifying an option or a user name, it lists the following information about all users on the local host

where you are logged in, if the information is in the `/etc/passwd` file for a given user:

- Login name
- Full name
- Terminal line name and whether it can receive messages from other users through `write` (see Section 11.8) or `talk` (see Section 11.9)
- Idle time
- Login time
- User's office location

10.2.1 Obtaining Information About a Specific User

If you specify the login name of a user on your local host, the `finger` command displays more information than if you entered the `finger` command without specifying a user name. The following additional information about the user is displayed:

- User's home directory and login shell
- Contents (if any) of the `.plan` and `.project` files in the user's home directory

The following example shows how to use the `finger` command to find information about user `smith`, who has an account on your local host:

```
% finger smith
Login name: smith      (messages off)  In real life: John Smith
Office: LV05-3/T24
Directory: /usr/netd/r2/smith          Shell: /bin/csh
On since Apr  9 16:20:56 on ttytb from wombat.lv5.dec.c
18 seconds Idle Time
Project: book, "Communicating with Network Users"
Plan:
Get information in the following areas:
  network commands
  mailx
```

In the first line of output, `messages off` means that user `smith` has put the `mesg n` command in his `.login` file to prevent his terminal from receiving messages from other users through the `write` or `talk` commands, which can be distracting.

The preceding example also displays the contents of the `.project` file and the `.plan` file that user `smith` created in his home directory. The `.project` file can contain only one line. The `.plan` file can contain as many lines as the file system allows; `finger` will print all the lines until the end-of-file (EOF) is reached.

10.2.2 Obtaining Information About Users on a Remote Host

In the following example, the `finger` command displays information about users on the remote host `boston`:

```
% finger @boston
[boston]
Login      Name          TTY Idle   When      Office
amy       Amy Wilson    p0    4 Thu 10:00 345
chang     Peter Chang   *p1  2:58 Thu 10:16 103
```

The first output line lists the remote host name, `boston`, and the second line describes the type of information in each column of the remaining output, each line allocated to one user. The asterisk (*) indicates that user `chang` has put the command, `mesg n` in his `.login` file to prevent his terminal from receiving messages from other users through the `write` or `talk` commands.

10.2.3 Obtaining Information About an Individual User on a Remote Host

To display information about user `luis` on remote host `havana` use the following `finger` command:

```
% finger luis@havana
Login name: luis                In real life: Luis Aguilera
Directory: /users/luis         Shell: /bin/csh
On since May 24 10:16:07 on ttyp2 from :0.0
58 minutes Idle Time
Project: baseball game simulation software
Plan:
Distribute with linked statistics module.
```

10.2.4 Customizing Output from the `finger` Command

There are several options to the `finger` command that enable you to modify the output according to the data you need. Table 10-1 lists and describes each option.

Table 10-1: Options to the `finger` Command

Option	Description
-b	Produces a brief version of output
-f	Suppresses display of titles of each field
-h	Suppresses printing of users' <code>.project</code> files

Table 10-1: (continued)

Option	Description
-i	Displays list of users with idle times
-l	Produces long format of output despite other options
-m	Assumes that <code>user</code> is an account name
-p	Suppresses printing of users' <code>.plan</code> files
-q	Displays only users' login and terminal names and login time
-s	Produces brief format of output despite other options
-w	Produces narrow, brief format of output despite other options

For more information on the `finger` command, see the `finger(1)` reference page.

10.3 Obtaining Information about Remote Hosts and Users

Before you send messages or transfer files over the network using the commands described in this book, you should know whether or not the recipient host is currently online. To do this, use the `ruptime` command which works for hosts that are running the `rwhod` daemon on the local network.

The `ruptime` command displays the following information:

- Host name
- Online status (`up` for online or `down` for offline)
- The length of time the host has been on line (or off line)
- The number of users currently logged in to the host (optionally including those whose sessions have been idle for an hour or longer)
- Load average statistics in 5-, 10-, or 15-minute intervals prior to the `ruptime` request

The syntax of the `ruptime` command is:

```
ruptime [ [ option... ] [ sort_option ] ]
```

If you use the `ruptime` command without options, a status report about the hosts on your local network, sorted alphabetically by host name, is displayed. For example:

```
% ruptime
apple      up 102+05:07   4 users,   load 0.09, 0.04, 0.04
byblos    up   3+03:17,  3 users,   load 0.08, 0.07, 0.04
carpal    up    2:28,    0 users,   load 7.01, 5.02, 3.03
dull      down  9+21:59
eager     down 23+22:45
foobar    up   3+01:44,  9 users,   load 0.01, 0.02, 0.03
garlic    up  14+01:35,  1 user,    load 0.06, 0.12, 0.11
hiccup    up   4+22:14, 19 users,   load 6.37, 3.90, 2.71
jackal    up  13+10:32, 26 users,   load 0.70, 0.92, 0.95
starry    up  16+21:08,  1 user,    load 0.22, 0.14, 0.07
travel    up  13+23:44,  7 users,   load 1.01, 1.19, 0.5
trekky    down 23+03:53
tribbl    up   8+21:43,  0 users,   load 0.00, 0.00, 0.00
trubbl    up  14+02:34,  0 users,   load 0.00, 0.00, 0.00
tunnel    down 14+02:34
warp9     up   8+01:24,  9 users,   load 0.01, 0.02, 0.03
```

Often, you need to determine only whether a single host is currently online. To do this, enter the `ruptime` command with the host name, as shown in the following example, for host `trekky`:

```
% ruptime trekky
trekky     down 23+03:53
```

This output shows that host `trekky` is not currently online.

You can also determine whether a host is online by using the `ping` command described in Section 10.5; `ping` works for any host in a TCP/IP network configuration.

If you plan to run commands on a remote host (as described in Chapter 13), use the `ruptime` command with the `-l` option to determine whether the host resources will be adequate. This command sorts the hosts by load average in descending order. The following example shows partial output from the `ruptime -l` command:

```
% ruptime -l
carpal    up    2:28,    0 users,   load 7.01, 5.02, 3.03
hiccup    up   4+22:14, 19 users,   load 6.37, 3.90, 2.71
travel    up  13+23:44,  7 users,   load 1.01, 1.19, 0.5
jackal    up  13+10:32, 26 users,   load 0.70, 0.92, 0.95
.
.
.
```

In this example, usage is low on all hosts except `carpal` and `hiccup`. Therefore, you may decide to remotely log in to either `travel` or `jackal`, if either host is suitable for your purpose.

If you need to use a remote host for a long period of time, you should know the total number of users there, not just the number whose sessions have been active for an hour or longer. Use the `ruptime` command with the `-a` option to display the total number of users on a remote host. The following two examples use the `ruptime -a` command to determine the total number of users first on host `travel`, and then on host `jackal`:

```
% ruptime -a travel
travel  up 13+23:44,    32 users,   load 1.01, 1.19, 0.5

% ruptime -a jackal
jackal  up 13+10:32,    29 users,   load 0.70, 0.92, 0.95
```

From the results of the `ruptime` command using the `-a` and `-l` options (in the preceding example), you can determine that both hosts have nearly the same number of users, but the current usage on host `travel` is calculated from only the 7 (from a total of 32) users whose sessions have been active for an hour or longer. By contrast, usage on host `jackal` is less, but is calculated from 26 of the total of 29 users. You could conclude that, over a period of time, usage on host `travel` may increase as more users log in, but that usage on host `jackal` may either decrease or stay nearly the same, because most of its users are already logged in.

The remaining options (except for `-r`) sort by different output fields, and in descending alphabetical order. To reverse this order, put the `-r` option after the other option on the command line. You should not combine other `ruptime` command options; if you do, only the last option on the command line will be used. Table 10-2 describes each option.

Table 10-2: Options to the `ruptime` Command

Option	Description
<code>-a</code>	Provides information for all users, including those whose sessions have been idle for an hour or longer
<code>-l</code>	Sorts output by load average over 5-, 10-, and 15-minute intervals
<code>-r</code>	Reverses the sort order
<code>-t</code>	Sorts output by length of time host is online
<code>-u</code>	Sorts output by number of users

For more information, see the `ruptime(1)` reference page.

10.4 Obtaining Information About Users on Remote Hosts

Before using a command that sends a message or transfers a file, you often need to know if the recipient user is logged in. To determine whether a user is logged in to a remote host on the local network, you can use the `rwho` command, specifying the name of one or more users. The `rwho` command operates only for hosts running the `rwhod` daemon. See your system administrator if necessary.

The `rwho` command displays the following information:

- User name
- Host name
- Start date and time
- Number of minutes a user's session has been inactive.

The `rwho` command has the following syntax:

```
rwho [ [-a] [ user... ] ]
```

Without options, the `rwho` command lists all users currently logged in to hosts on the local network, except those who have been idle for an hour or longer. A typical local network has several dozen users, so you should specify only the users about whom you need information.

Although the `-a` option displays all users, including those idle for more than an hour, you can still use it while specifying only certain users. This enables you to determine whether or not a remote user is logged in, regardless of whether that user has been inactive for an hour or longer. The following example uses `rwho` with the `-a` option to determine this information for users `wally`, `becky`, and `smith`:

```
% rwho -a wally becky smith
becky  cygnus:pts0      Jan 17 11:20 :12
smith  aquila:ttyp0         Jan 15 09:52 :22
wally  lyra:pts7            Jan 17 13:15 1:32
wally  lyra:pts8            Jan 17 14:15 1:01
```

As shown, the output from the `rwho` command displays in alphabetical order by user name, then by host name. The amount of idle time greater than one hour is shown in the last column, after the starting time and date of each session. Without the `-a` option, the information for user `wally` would not have displayed.

For more information on the `rwho` command, see the `rwho(1)` reference page.

10.5 Determining Whether a Remote Host Is Online

The `ping` command is used by system administrators to fix network transmission problems and works for any host configured in a TCP/IP network. As a network user, you can use it to determine whether a remote host is currently online. For example, to determine whether remote host `moon` is online, enter the `ping` command at your local system prompt. The output, which verifies that the remote host is online, will continue to display until you press `Ctrl/c`, as shown in the following example:

```
% ping moon
PING moon (130.180.4.108): 56 data bytes
64 bytes from 130.180.4.108: icmp_seq=0 ttl=255 time=42 ms
64 bytes from 130.180.4.108: icmp_seq=1 ttl=255 time=0 ms
64 bytes from 130.180.4.108: icmp_seq=2 ttl=255 time=0 ms
64 bytes from 130.180.4.108: icmp_seq=3 ttl=255 time=0 ms
\*(KbCtrl/c\*(Ke
----moon PING Statistics----
9 packets transmitted, 9 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 0/4/42 ms
```


Sending and Receiving Messages 11

This chapter describes how to send or receive messages over the network by using one of the following commands:

- `mailx` or `Mail`
- `write`
- Message Handling (MH) program
- `talk`

Examples in this chapter use the `mailx` program rather than `Mail`. Using `mailx`, you can do the following tasks:

- Send a message to a user
- Edit a message before sending it
- Include files within messages
- Save or organize incoming messages

Using `mailx` you can also send entire files, a task described in Chapter 12, "Copying Files to Another Host."

The Digital UNIX `mailx` or `Mail` command accesses the same mail program as the ULTRIX `mail` (that is, `/usr/ucb/mail`) command. Refer to Chapter 2 of the *ULTRIX to Digital UNIX Migration Guide* for more information.

The `write` and `talk` commands work interactively; the recipient must be logged in. Before using these interactive commands, you can verify the name and availability of a user or host by using the following commands described in Chapter 10:

- `finger` or `who` to find a user on the local host
- `finger`, `rhwo`, or `ruptime` to find a user on a remote host
- `ping` or `ruptime` to find a currently reachable host

11.1 Addressing Mail Messages

Using `mailx`, you can send a message to one or more users at the following locations:

- On your local host
- On a remote host connected to your local host via TCP/IP
- On a host in another network, through either TCP/IP, DECnet, or UUCP addressing

Use the following syntax for the `mailx` command:

```
mailx user[ @{ host | domain | host.domain } ] ...
```

To send mail to users on the local host, enter the `mailx` command and specify a `user` parameter for each user. For users on remote hosts, you must specify additional information about the location of the host after the “at sign” (@).

For example, to send mail from host `orange` to users `smith` and `jones` on the same local host, you would enter the following command:

```
orange% mailx smith jones
```

To send mail to user `hobbes` on a different host, `pluto`, in the same **domain**, you would enter the following command:

```
orange% mailx hobbes@pluto
```

In the preceding example, if user `hobbes` were in another domain called `planets`, you would add the name of the remote network domain, as shown in the following command:

```
orange% mailx hobbes@pluto.planets
```

The domain is sometimes split into further subdivisions with the name of each separated by a period (.) in the destination name. Depending on how the network has been configured by the network administrator, you can address a user on a remote host in another domain by specifying only the domain name, as in the following command:

```
orange% mailx hobbes@planets
```

If necessary, see your system administrator for help addressing a mail message.

11.2 Sending a Mail Message Using `mailx`

This section explains how to use `mailx` to send a message to a user on a local host and a copy of the message to other users.

To begin the example, user Jones enters the `mailx` command at the local system prompt, `orange%`:

```
orange% mailx suzuki Return
```

After pressing the Return key, the `Subject:` prompt displays. Pressing the Return key again immediately would leave the subject blank. Instead, user Jones enters the subject of the message before pressing the Return key, and then begins writing the message:

```
Subject: Baseball question Return
Are there any Japanese baseball simulation games?
I want to compare Sadharu Oh's hitting statistics
to those of Hank Aaron. To do this, I need to set
up a simulated baseball season having each hitter
play for one year in the other player's league.
```

User Jones ends the message by typing a period (.) on a blank line, followed by the Return key, as shown:

```
.
.
.
play for one year in the other player's league.
. Return
Cc:
```

In this example, the message has not yet been sent; instead the `Cc:` (that is, “carbon copy”) prompt appears because user Jones has customized his mail session by adding `set askcc` to the `.mailrc` file in his home directory. (See Section 11.6 and Appendix D on customizing your mail session.)

The `Cc:` prompt enables you to send copies to other users. If you choose not to, press the Return key to exit `mailx`, and send the message; the end-of-text message (EOT), then appears followed by the system prompt.

In this example, user Jones sends copies to local user `cranton` and remote users `gillis` and `vincep` by entering the appropriate address for each at the `Cc:` prompt and pressing the Return key to exit `mailx` and send the message:

```
.
.
.
Cc: cranton gillis@strato vincep@mlb.bbs.com Return
EOT
orange%
```

The `mailx` program enables you to recover from addressing errors. For example, if your intended recipient on the local host is `cranton`, but you mistakenly type `crantom`, the following message appears immediately on your screen,

```
crantom... User unknown
```

and the message is mailed back to you. You can then save and resend it to the right person.

If you send a message to an unknown person on a remote host, it may take as long as three days before `mailx` sends it back to you. Section 11.3.3.1 explains how to save and resend a returned message.

11.2.1 Editing a Message

To edit a mail message before sending it, after replying to the `Subject :` prompt, enter one of the following **escape** commands to activate an editor within `mailx`:

- Enter `~v` to activate the screen editor that you set with the `set VISUAL` entry in your `.mailrc` file.
- Enter `~e` to activate the text editor that you set with the `set EDITOR` entry in your `.mailrc` file.

To use the `~e` command from within `mailx` to activate a text editor, enter `~e` as the first two characters on a new line – you may need to type the tilde (`~`) a few times before it displays. For example:

```
Subject: network documentation meeting at 2 PM
Everyone, please bring the Table of Contents
for your book so that we can look for areas
of overlapping subject matter and
~e
```

If your `.mailrc` file contains `set EDITOR=/usr/ucb/vi`, you can now use the `vi` editor to correct the spelling mistake in the first line and finish writing the message. When you end the editing session, you are back in `mailx`. You can end the message and exit `mailx` or re-invoke `vi` and continue writing.

11.2.2 Aborting a Message

You may decide not to send a message that you have started. There are two ways to abort a message before sending it.

11.2.2.1 Aborting a Message with Ctrl/c

You can abort a mail message by pressing Ctrl/c twice, anywhere within a message. When you first press Ctrl/c, the following message is displayed:

```
(Interrupt -- one more to kill letter)
```

You can now reconsider your decision to abort the message. If you decide not to abort it, continue entering text. If you decide to abort the message, press Ctrl/c again, and the following message will be displayed:

```
(Last Interrupt -- letter saved in dead.letter)
```

The message is aborted, you exit mailx, and the system prompt is displayed.

By default, the aborted message is saved in the `dead.letter` file in your home directory. If you choose not to save aborted messages, you can enter `set nosave` in your `.mailrc` file. See Section 11.6 and Appendix D for more information.

Only the most-recently aborted message is saved in the `dead.letter` file. You can edit and resend it by including it within a mail message. (See Section 11.2.3 for information on including files within messages.)

The following example shows how to abort a mail message by pressing Ctrl/c:

```
orange% mailx sally
Subject: Update to reference page files
What should the mailx(1) reference page include
about sending to remote users? Ctrl/C
(Interrupt -- one more to kill letter)
Ctrl/C
(Last Interrupt -- letter saved in dead.letter)
orange%
```

11.2.2.2 Aborting a Message with an Escape Command

You can abort a mail message by entering either the `~q` or the `~x` escape command on a blank line. Unlike aborting a message by pressing Ctrl/c, these commands abort the message immediately, without prompting you to reconsider. The `~q` escape command saves the aborted text in the `dead.letter` file in your home directory, but `~x` does not, even if you have `set save` in your `.mailrc` file.

The following example shows how to abort a mail message by using the `~x` escape command:

```
orange% mailx sally
Subject: Update to reference page files
What should the mailx(1) reference page include
~x
orange%
```

You may need to enter the beginning tilde character (`~`) a few times before it appears.

11.2.3 Including a File within a Message

You can include any file (except an unconverted binary file) within a mail message. You will do this often when you save and resend any incorrectly-addressed mail that is returned to you (See Section 11.3.3.1) or when you edit and resend an aborted message saved in the `dead.letter` file in your home directory.

From an example in the previous section, the `dead.letter` file contains the following text:

```
What should the mailx(1) reference page include
about sending to remote users?
```

Suppose that you want to resend this file to user `sally` after adding additional information. While in `mailx`, use the `~d` escape command to automatically add the text of `dead.letter` to the mail message, regardless of the current working directory.

The following example starts a message to user `sally` before adding the text of `dead.letter` through the `~d` command:

```
orange% mailx sally
Subject: the mailx(1) reference page
The uucp(1) reference page has formatting
information for sending to remote users.
~d
"/usr/staff/r2/sally/dead.letter" 2/76
```

After including the file, its full path name is displayed, with the number of lines (2) and characters (76) that the file contains (including the Return key or a control character at the end of each line). After the display, you can exit or continue writing your message, but you may want to enter a text editor (for example, by typing `~v` for `vi`) to look at the included file, which is not displayed otherwise.

Note

The `dead.letter` file contains only the most-recently aborted mail message. You may want to verify that it contains the text that you want to send.

To include a file (including `dead.letter`) within a mail message, you can use either of two escape commands, `~<` or `~r` followed by the file name. These commands work in the same way. If the file is not in the same directory from which you entered `mailx`, you must precede the file name with a full path name or one that is relative to the current directory.

The following example uses the `~<` escape command to include a file called `strato_prob` from the `environ` directory below the current working directory:

```
orange%  
mailx sally  
Subject: Dan, here's the stratosphere data file  
~< environ/strato_prob  
"environ/strato_prob" 41/1309
```

See Section 12.1.3 for information about transferring a file non-interactively through the `mailx` utility.

11.3 Receiving a Mail Message

When you receive a mail message, you have the following options:

- Delete or read any message waiting for you.
- Reply to the sender and any other recipients to whom the mail was sent.
- Save the message in a file.
- Organize the message by topic in a file of saved messages called a **folder**.

The `mailx` program notifies you if you have new mail when you log in, enter any operating system command, or press the Return key. You can also enter the `mailx` command at the system prompt to see if you have new mail.

In the following example, user Jones on host `orange` enters `mailx` and finds two messages waiting:

```
orange% mailx  
Mail $Revision: 1.1.9.4 $ Type ? for help.  
"/usr/spool/mail/jones": 2 messages 1 new 1 unread  
U 1 root Mon Jul 20 10:39 14/438 "System news"  
>N 2 root Mon Jul 20 11:30 11/292 "Welcome"  
?
```

In this example, two messages are waiting from the system administrator (root); one is unread (denoted by U in column 1) from a previous mailx session and the other is new (denoted by N). The question-mark (?) at the end of the message is the mailx prompt. You can type another question-mark at this prompt to display a list of available mailx commands, as indicated in the header and as described in Section 11.4.

You can press the Return key at the mailx prompt to read message 2, which is indicated by the right-angle bracket (>), in the list of waiting messages. For example,

```
? Return
Message 2:
From root Wed Aug 4 11:17:36 1996
Date: Wed, 4 Aug 1996 11:17:29 -0400
From: root (system administrator)
To: jones
Subject: Welcome
```

```
Welcome to the company computer network. I'm the
person who manages this system. If you have
questions or problems, send mail to root. You
can also send mail to manager or admin; messages
will be forwarded to me.
```

```
I will be on vacation for the next two weeks after
this week... starting Monday, August 10. I'll be
stdin Space
back on Monday, August 24.
?
```

In this example, `stdin` displays because the `PAGER` variable has been set to `more` (the default). If the `PAGER` variable had been set to `pg` nothing would have been displayed. Also in the above example `stdin` appears after the 15th line of the message because user Jones has customized the mailx environment by adding `set crt=15` to the `.mailrc` file. In the `.mailrc` file, `set crt=` specifies the number of lines to display at one time before invoking the pager (either `pg` or `more`). To display the remainder of the message. As shown in the example, because the message is more than 15 lines long, `set crt` instructs mailx to invoke the pager after 15 lines. By pressing the Space bar, the next screen full of the message are displayed. You should customize your mailx environment by using `set crt=`. Otherwise, long mail messages will scroll rapidly, requiring you to quickly press the Hold Screen key. See Section 11.6.2 and Appendix D for more information on customizing mailx.

To read another message, enter the message number at the `mailx` prompt. To list the messages again, enter `h` at the `mailx` prompt. In the following example, user Jones uses the `h` command to list the mail messages, sees that the first message is still unread, and enters a `1` at the `?` prompt to read it:

```
? h
  U 1 root   Mon Jul 20 10:39 14/438  "System news"
>  2 root   Mon Jul 20 11:30 11/292  "Welcome"
? 1
Message 1:
From root Mon Jul 20 11:30:07 1996
Date: Mon, 20 Jul 1996 11:30:04 -0400
From: root (system administrator)
To: jones
Subject: System news
```

```
The newest release of the text processing
software will be installed after 5 o'clock
today. Send mail if you have questions or
concerns before or after the installation.
```

```
?
```

The message you are reading is called the **current message**. To reread it, enter a period (`.`) at the `mailx` prompt. To read the next message, press the Return key. This message becomes the current message. You can read all your messages in succession by pressing the Return key after each message.

11.3.1 Deleting a Message

Your messages stay in `mailx` until you delete them before or after storing them in a file or in a folder. To delete the current message after reading it, enter the `d` (delete) command at the `mailx` prompt. To delete a different message, enter the `d` command at the `mailx` prompt, followed by the message number. You can delete several messages by listing their numbers after the `d` command. For example, enter the following command at the `mailx` prompt to delete messages 7 and 9:

```
? d 7 9
```

You can also delete a range of messages by using a hyphen between the first and last message. For example, to delete messages 7 through 11, enter the following command at the `mailx` prompt:

```
? d 7-11
```

If you accidentally delete a message, you can recover it with the `u` (undelete) command. For example, to undelete message 7, enter the following command at the `mailx` prompt:

```
? u 7
```

If you exit `mailx` by entering `q` or `quite` instead of `x`, any previously read messages that you do not delete are added to the end of a file of previously undeleted messages named `mbox`, in your home directory.

11.3.2 Replying to a Message

Replying to a mail message is similar to sending a mail message. You have the same options to edit, abort, or include a file in a message, as described in Section 11.2.

To reply to the sender of a message that you have just finished reading, enter an uppercase `R` (reply) command at the mail prompt, as shown in the following example:

```
Message 3:
From: deedee Mon Jul 20 14:13:32 1996
Date: Mon, 20 Jul 1996 14:13:05 -0400
From: deedee (DeeDee Smith)
To: jones, mays@sf24.usenet, susannah@artwrk
Subject: Testing text-processing software
```

```
I think we should test the new text processing
software on the older machines as well as the
newer. Remember that many customers still have
the older models.
```

```
? R
To: deedee
Subject: Re: Testing text-processing software
```

```
I agree. Also, we should test different machine
configurations to determine if, for example,
it performs satisfactorily when run remotely.
```

```
.
EOT
?
```

After you enter `R`, the recipient and subject line display, enabling you to verify that you are replying to the intended recipient.

If you enter a lowercase `r`, the reply will be sent to the recipients of the original mail as well as to the sender; in this example, to `mays@sf24.usenet` and `susannah@artwrk`, as well as to `deedee`.

Note

To reply only to the sender of a mail message, enter an uppercase `R` at the mail prompt. To reply to the sender and all recipients of a mail message, enter a lowercase `r` at the prompt.

In the preceding example, the `Cc:` prompt does not appear because user `jones'` `.mailrc` file does not contain the `set askcc` command.

11.3.3 Saving a Message

If you leave `mailx` by entering the `q` command (instead of the `x` or `exit` command), the messages that you have just read are stored in the `mbox` file in your home directory. You can read this file by using the `more` or `cat` command, or you can use a text editor to read and edit it.

To store mail messages in a more useful way, you can save them in individually-named files or in folders as described in the following sections.

11.3.3.1 Saving a Message in a File

There are several kinds of mail that you might want to save in a file:

- A brief but important message that you read
- An incorrectly-addressed mail message that `mailx` returns to you
- A long message that you want to print and read later

To save a brief message that you read or a mail message that `mailx` returned to you, enter the `s` command at the `mailx` prompt and supply a name for the file.

A returned message is shown in the second item in the following output from the `mailx` command:

```
orange% mailx
Mail $Revision: 1.1.9.4 $ Type ? for help.
"/usr/spool/mail/jones": 2 messages 1 new 1 unread
  U 1 root    Mon Jul 20 10:39 14/438  "System news"
>N 2 MAILER-DAEMON Wed Aug  5 09:39 19/498 "Returned mail: User
unknown"
?
```

As shown below, user Jones decides to save the returned message from the previous example in the file, `verify-resend`, as a reminder to find the

correct address before resending it.

```
.  
.  
.  
>N 2 MAILER-DAEMON Wed Aug 5 09:39 19/498 "Returned mail: User  
unknown"  
? s verify-resend
```

The file, `verify-resend`, is saved in the current directory unless an explicit pathname is specified. For example, user Jones could have saved it in a subdirectory called `fix-later` by entering the following command:

```
? s fix-later/verify-resend
```

To save a long message without reading the entire text online, press `Ctrl/c` to stop the message from scrolling and to display the `mailx` prompt. You can now save (or delete) the message. In the following example, user Jones receives a 20-page report in message 1, and presses `Ctrl/c` to access the `mailx` prompt where a command is entered for saving the file:

```
? 1  
Message 1:  
From smith Wed Aug 5 16:43:42 1996  
Date: Wed, 5 Aug 1996 16:43:41 -0400  
From: smith (Cassandra Smith)  
To: jones  
Subject: 20-page report: host configuration results
```

Mortimer,

Here's the report on host configuration that the
`Ctrl/c`

Interrupt

```
?s sys-config-report  
"sys-config-report" [New file] 2147/48353  
?dp  
.  
.  
.
```

In the example above, after creating the file, user Jones enters `dp` (that is, delete-proceed) at the `mailx` prompt to prevent the large 20-page file from being saved in `mbox`, and to start reading the next mail message. Otherwise, the `mailx` command, `d` (delete) and pressing the Return key could have been entered to do the same.

11.3.3.2 Saving a Message in a Folder

To organize messages for easier reference and to minimize the size of the `mbox` file (which is a folder itself) you can save messages in files called **folders**. Before using a folder other than `mbox`, you must create a folder sub-directory in your home directory and add a pointer to the folder in your `.mailrc` file. For example, if you make a directory named `folder` in

your home directory, you must add the following line to your `.mailrc` file:
`set folder=folder.`

To add a message to a folder, use the `mailx` command, `s` and the folder name. For example, to save a message about host configuration with other messages on that topic, write it to a folder named `sys-config`, as follows:

```
Message 7:
From smith Thu Aug  6 09:32:09 1996
Date: Thu, 6 Aug 1996 09:32:08 -0400
From: smith (Cassandra Smith)
To: jones
Subject: host configuration testing
```

```
According to the report that each LAN ...
? s sys-config
"sys-config [New file] 11/235
```

When you save the first message in a folder, `mailx` stores it and displays the message, `New file`. If you save more messages in that folder, `mailx` appends them to the end of the file and displays the message, `Appended`.

There are two ways to read messages stored in a folder other than `mbox`:

- From the shell prompt you can start `mailx` with the `-f` option and the folder name. For example, to read the `sys-config` folder, enter the following command:

```
orange% mailx -f sys-config
```

- If you are already in `mailx`, use the `folder` command to switch to a different folder. For example, if you are reading the `sys-config` folder and you want to read the `meetings` folder, enter the following command:

```
? folder meetings
```

When you switch folders, `mailx` makes any changes to the folder you are leaving before it opens the new folder.

You can use the `folder` command without arguments to find out what folder you are in. For example:

```
? folder
"sys-config": 17 messages
```

11.3.4 Forwarding a Message

You can use the `m` command and the `~f` escape command in `mailx` to forward to other users any message that you receive. The following example shows how to start message forwarding to user `deedee`:

```
? m deedee
Subject: forwarding a message
```

For example, to forward message number 3 to user `deedee` and include a subject line and introductory note, you would enter the following text:

```
I received this note from Gary. Do you agree?
~f 3
Interpolating: 3
(continue)
.
EOT
```

As shown, after you enter the `~f` command, you can continue writing or end the message. To forward the current message, do not enter a number after the `~f`.

11.4 Getting Help from mailx

When you enter `mailx` and messages are waiting, the following line is displayed at the top of the header:

```
Mail $Revision: 1.1.9.4 $ Type ? for help.
```

This is a reminder that you can type a question mark (?) at the `mailx` prompt to display a brief description of available `mailx` commands, as shown in Example 11-1.

Example 11-1: Output from mailx Help Command

```
? ?
Control Commands:
  q          Quit - apply mailbox commands entered this session.
  x          Quit - restore mailbox to original state.
  ! <cmd>    Start a shell, run <cmd>, and return to mailbox.
  cd <dir>]  Change directory to <dir> or $HOME.
Display Commands:
  t [<msg_list>] Display messages in <msg_list> or current message.
  n          Display next message.
  f [<msg_list>] Display headings of messages.
  h [<num>]   Display headings of group containing message <num>.
Message Handling:
  e [<num>]   Edit message <num> (default editor is e).
  d [<msg_list>] Delete messages in <msg_list> or current message.
  u [<msg_list>] Recall deleted messages.
  s [<msg_list>] <file> Append messages (with headings) to <file>.
  w [<msg_list>] <file> Append messages (text only) to <file>.
  pre [<msg_list>] Keep messages in system mailbox.
Creating New Mail:
```

Example 11-1: (continued)

```
m <addrlist>    Create/send new message to addresses in <addrlist>.
r [<msg_list>]  Send reply to senders and recipients of messages.
R [<msg_list>]  Send reply only to senders of messages.
a              Display list of aliases and their addresses.
===== Mailbox Commands =====
```

11.5 Exiting Mail

There are three commands you can use to exit from `mailx`:

- The `q` command returns you to the shell prompt and saves in the `mbox` file in your home directory, any messages you read but did not delete.
- The `x` and `exit` commands are the same. Each returns you to the shell prompt without changing your mailbox.

11.6 Customizing Mail Sessions

When setting up an account for a new user, the system manager defines certain `mailx` default settings in the `/usr/share/lib/Mail.rc` file. As supplied by the operating system, this file contains the following `mailx` settings, which a user can override:

- The `set ask` setting activates the `Subject:` prompt.
- The `set noaskcc` setting deactivates the `Cc:` prompt.
- The `set dot` setting means that a single dot on a blank line (`.`) terminates the mail message.
- The `set nokeep` setting means that the system mailbox is deleted when it becomes empty. This setting is unimportant to most users.
- The `set save` setting means that aborted messages are saved in the `dead.letter` file.

You can customize your `mailx` session by defining aliases and setting variables in the `.mailrc` file in your home directory. Here is a sample `.mailrc` file:

```
alias sue susannah
alias wombats tom, jeff, craig, jim, ken
set ask
set askcc
set prompt=>
unset dot
set record=/usr/users/hale/outgoing
set folder=folder
set crt=20
```

11.6.1 Creating Mail Aliases

You can use the `alias` command in `mailx` to create alternate names for users or user groups.

Note

The `mailx` alias is not the same `alias` command used by the shell; you cannot use it to modify mail commands.

To define a permanent mail alias, enter the `alias` command in the `.mailrc` file, specifying the alias name and one or more login names. The following `.mailrc` file defines two aliases:

```
alias sue susannah
alias wombats tom, jeff, craig, jim, ken
```

The first alias defines the name `sue` to mean user `susannah`. This enables you to send mail to `susannah` by using the name `sue`. The second alias enables you to send mail to members of a team called the Wombats – `tom`, `jeff`, `craig`, `jim`, and `ken`, by addressing your message to `wombats`. Another way to enter this line in `.mailrc` is this:

```
alias wombats tom,\
                jeff,\
                craig,\
                jim,\
                ken
```

The backslash (`\`) enables you to write a single long command on several lines.

While in `mailx`, you can see what aliases are defined by using the `alias` command without arguments. You can also define temporary aliases at the `mailx` prompt that are in effect during that `mailx` session.

11.6.2 Setting Mail Variables

Mail variables are similar to variables in your `.login` file. They can be binary, string, or numeric.

To set a binary mail variable in your `.mailrc` file, enter the `set` command followed by the option name. The sample `.mailrc` file includes these binary variables:

```
set ask
set askcc
unset dot
```

- Setting `ask` makes `mailx` prompt you for the subject line of messages you send.

- Setting `askcc` makes `mailx` prompt you for carbon-copy recipients.
- Unsetting `dot` makes `mailx` refuse to end a message when you type a line with just a period on it; you would have to end a message by pressing `Ctrl/d` instead.

String mail variables accept characters or numbers as values. The sample `.mailrc` file includes the following three string variables:

```
set folder=folder
set record=/usr/users/hale/folder/outgoing
set crt=20
```

- The `folder` variable defines a subdirectory to contain your mail folders. If you set this variable, the `mailx` utility creates folders as files in this directory when you save messages using the `save folder` command. The `mailx` utility interprets the file name as a subdirectory of your home directory.
- The `record` variable tells `mailx` to put a copy of each message you send in the file you specify. If you do not set this variable, no automatic record is kept. In this example, we've specified a file that will be treated as an ordinary folder by `mailx`. To select the record file, use the following command:


```
orange% mail -f outgoing
```
- The `crt` variable tells `mailx` how many lines of a message should be displayed before invoking the pager program.

11.7 The Message Handling (MH) Program

An alternative to the `mailx` program is the Message Handling program (MH). The MH program is a set of small mail-handling programs that you use by entering the command you want to execute from the shell prompt.

The MH program is optional; it may not be installed on your host. To determine if MH is available, look for the `/usr/bin/mh` directory.

To use MH, you must add the `/usr/bin/mh` directory to your path by editing the `set path` line in your `.cshrc` or `.login` file. Then, notify the shell about the change in your path by logging out and logging back in, or by entering the following command (for the C shell):

```
orange% source .login
```

If your path is set in `.cshrc`, use `.cshrc` instead of `.login` in this command.

For either the Bourne or Korn shell, you would add this information to the `.profile` file and notify the shell by entering the following command:

```
orange$ . .profile
```

With the MH program, folders are organized differently from `mailx` folders. New and unread mail is kept in a folder called `inbox`, into which you move the mail that arrives in your system mailbox by using the `inc` command. You must enter the `inc` command every time you want to include new mail.

You select a folder with the `folder` command. If you enter it without a folder name, `folder` displays the currently selected folder.

You can enter the `folder` command with the `-all` option to display a list of your folders and the number of messages in each.

You use the `show`, `prev`, and `next` commands to read the current, previous, and next messages in your current folder. If you enter a message number with the `show` command, that message becomes your current message. For example:

```
orange% show 7
Message 7:
From deedee Mon Jul 23 10:02:10 1996
Date: Mon, 23 Jul 96 10:01:25 edt
To: hale
Subject: Cafeteria hours
Cc:
Status: R
```

```
I'm sorry you didn't ask that sooner. The cafeteria
closes its breakfast service at 10. Lunch starts
at 11:30.
```

The `rmm` command removes messages from your current folder. If you use the `rmm` command with no argument, it deletes the current message. If you specify one or more message numbers, the messages you specify are removed. For example, to remove messages 2, 5, and 7, enter the following command:

```
orange% rmm 2 5 7
```

Table 11-1 lists most of the MH commands. For a complete list, see the `mh(1)` reference page. For more information about each MH command, see the reference page for each.

Table 11-1: Commands for the MH Message-Handling Program

Command	Description
<code>ali</code>	Searches the specified alias files and displays the addresses corresponding to the specified aliases.
<code>anno</code>	Annotates messages to keep track of distribution, forwarding, and replies for your messages.
<code>burst</code>	Extracts the original messages from a forwarded message, discards the forwarding header, and places the original messages at the end of the current folder.
<code>comp</code>	Creates a new mail message, providing a template for you to fill in and invoking an editor to finish the message.
<code>dist</code>	Redistributes the current message to addresses that are not on its original distribution list.
<code>folder</code>	Selects a folder or displays the contents of your current folder.
<code>folders</code>	Lists all your folders and the number of messages each one contains.
<code>forw</code>	Forwards messages to recipients who were not the original addressees. The message is encapsulated (included with a “Forwarded message” notice) and a header is added.
<code>inc</code>	Incorporates mail from your system mailbox into your <code>inbox</code> folder.
<code>mark</code>	Assigns a name to a sequence of messages in your current folder. You can then use the <code>pick</code> command to select messages marked in this way.
<code>mhl</code>	Lists formatted MH messages. You can use this command as a replacement for the <code>more</code> command to display messages.
<code>mhmail</code>	Sends mail to the specified users. If you do not specify any users, <code>mhmail</code> works like the <code>inc</code> command.
<code>msgchk</code>	Checks your system mailbox and any other files that can receive new mail for you, looking for new messages. If any new messages are found, <code>msgchk</code> reports.
<code>next</code>	Displays the next message in the current folder or in the specified folder.
<code>packf</code>	Compresses a folder into a single file. (Each message is normally stored as a separate file.) Do not confuse the <code>packf</code> command with the <code>pack</code> command.
<code>pick</code>	Selects messages based on content, sequence name, or other criteria.

Table 11-1: (continued)

Command	Description
<code>prev</code>	Displays the previous message in the current folder.
<code>prompter</code>	Invokes a simple editor designed for composing messages. The <code>prompter</code> command is invoked by <code>comp</code> , <code>dist</code> , <code>forw</code> , and <code>repl</code> ; you do not need to call <code>prompter</code> directly.
<code>rcvstore</code>	Incorporates a message from the standard input directly into a folder.
<code>refile</code>	Moves messages from the current folder to one or more other folders.
<code>repl</code>	Replies to either the current message or the message you specify.
<code>rmf</code>	Removes all of the messages in a folder and then removes the folder itself.
<code>rmm</code>	Removes messages from a folder. The message files are not actually destroyed; instead, <code>rmm</code> renames them by inserting a number sign (#) as the first character of the file names. On most hosts, files whose names begin with a number sign are deleted once a day by an automatic process. Until they are actually deleted, you can recover removed messages by using the <code>mv</code> command to rename the files.
<code>scan</code>	Displays a list of the messages in a folder.
<code>send</code>	Sends a message that you have created by using <code>comp</code> , <code>prompter</code> , or another editor.
<code>show</code>	Displays the contents of a message.
<code>sortm</code>	Sorts messages in a folder into chronological order according to the <code>Date:</code> field of the message header.
<code>whatnow</code>	Prompts you for what to do with a message you have just composed. You can reexamine an original message to which you are replying, resume editing the new message, or do other tasks associated with sending the message.
<code>whom</code>	Expands the header of a message into a set of addresses and optionally checks to see that the message can be delivered to those addresses.

The following example shows how the MH `msgchk` command reports new messages:

```
orange% msgchk
You have new mail waiting, last read on date
```

You can tailor the features of MH by creating a `.mh_profile` file in your top-level directory. The MH reference pages describe the features that you can modify.

11.8 Sending and Receiving Messages With `write`

The `write` command enables two users on the same or different hosts to communicate on either a video display terminal or on nonvideo devices (for example, a teletypewriter) that print messages on paper.

You can use `write` to send a message immediately to someone who you cannot reach by telephone, especially if you do not require a reply. (See also the `talk` command in Section 11.9.)

The `write` command displays a message on the terminal screen of the recipient. You can prevent users from communicating with you through `write` and `talk` by entering the `msg n` command in the `.login` file in your home directory. You cannot disable incoming messages from those with superuser privileges.

To determine whether a user on a local host has disabled messages from `write` and `talk`, use the `finger` command and look at the first line of output for the phrase `messages off`. For example:

```
Login name: smith   (messages off)  In real life: John Smith
.
.
.
```

For users on a remote host, the disabling of `write` and `talk` is denoted by an asterisk (*) in the TTY field of the output line, for example:

```
.
.
.
Login      Name           TTY Idle   When      Office
chang     Peter Chang   *p1 2:58 Thu 10:16  103
```

See Section 10.2.2 and `finger(1)` for more information.

Your intended recipient may be running a command that temporarily disables `write` to prevent its interference, and the sender would receive the message,

```
Write: Permission denied
```

just as if the recipient had explicitly disabled `write`.

You can use the `write` command only when the recipient is logged in. Use the `who` command, as described in Section 10.1, to list current users. If, for example, user `smith` is not logged on when you send a message through `write`, the following message is displayed on your terminal screen:

```
smith is not logged on
```

The following steps show how user `wang` sends a message to user `chung`, both of whom are logged in on local host `dancer`:

1. User `wang` enters the `write` command at the system prompt:

```
dancer% write chung
```

The `write` program rings a bell and sends the following message to the terminal screen of `chung`:

```
message from wang tty04 Feb 14 10:32:45
```

A bell rings user `wang`'s terminal when the connection is made.

2. User `wang` types the message, pressing the Return key after each line, and ends the message by pressing Ctrl/d. For example, `wang` sends the following message in two lines to user `chung`:

```
The double-sided lab printer is working. Return  
Re-send your job, and I'll check it. Return  
Ctrl/d
```

3. After `wang` presses Ctrl/d, the EOF (end of file) signal is displayed on the screen of user `chung` to indicate the end of the message.

Note

See your system administrator if pressing Ctrl/d does not produce the EOF signal on the recipient's screen or if a bell does not ring on the sender's terminal.

You can use the exclamation point (!) at the beginning of a new message line to access the shell prompt and execute any operating system command (including `write`). For example, if `wang` forgot the name of the current directory from which `chung` is to retrieve certain files, `wang` can enter the `!pwd` command to remind himself, as shown:

```
dancer% write chung
You can copy the network user files from: Return
!pwd Return
/ufs/usr/staff/r0/net-dir/network_comm
!
/ufs/usr/staff/r0/net-dir/network_comm
Ctrl/d
dancer%
```

The `write` command can be used interactively, but it is difficult for both sender and receiver to determine when the other has finished and is waiting for a reply. For example, `wang` can enter the following command:

```
dancer% write chung
```

`Wang` will then wait for `chung` to reply, but `chung` might also wait, thinking that `wang` intends to continue the message.

To minimize problems, it is a good idea to establish a simple, temporary protocol each time you want to use `write` interactively. For example, user `wang` can start his message to `chung` as follows:

```
dancer% write chung
I'll mark the end of each message with 'ZZZ' Return
and wait for a reply. Please do the same. Return
I'll install a driver for the new printer. Return
Do you want to test it? ZZZ Return
```

For more information, see the `write(1)` reference page.

11.9 Sending and Receiving Messages with `talk`

The `talk` command enables a user to send a message to another user on the same or on a remote host, interactively and more easily than through `write`. However, `talk` works only on video display terminals.

Like `write`, you can use `talk` to send a message immediately to someone who you cannot reach by telephone. Also, like `write`, the `talk` command may disrupt the receiver because it sends a notification message directly to a terminal and continues doing so until a reply is entered.

To disable incoming messages (except from those with superuser privileges) from `talk` (and from `write`, as described earlier) you can put the command, `mesg n` in your `.login` file. To determine whether a user has done this, use the `finger` command as described in Section 10.2.1 or in the `finger(1)` reference page.

During an online `talk` session, a “send window” and a “receive window” are opened on each user’s terminal. Each can type into the send window while `talk` displays in the receive window what the other user is typing.

For example, to send a message to user `hoover` on the same local host `apple`, user `coolidge` enters the following `talk` command:

```
apple% talk hoover
```

The program then divides the terminal screen of `coolidge` into two parts; the top half assigned to `coolidge` and the bottom half assigned to `hoover`.

Next, the following message is displayed in the top of the screen:

```
[No connection yet]
```

When the connection is established, the following message is displayed:

```
[Waiting for your party to respond]
```

After this message, a bell rings on `hoover`’s terminal and the following message is displayed:

```
Message from Talk_Daemon@apple at 16:18 ...
talk: connection requested by coolidge@apple
talk: respond with: talk coolidge@apple
```

If `hoover` does not respond quickly, the following message is displayed on `coolidge`’s screen:

```
[Ringing your party again]
```

When `hoover` responds, a message about the established connection appears on `coolidge`’s screen. Each user can now enter text. If the screen fills up, `talk` overwrites the text at the beginning of the screen. Either user can end the conversation by pressing `Ctrl/c`. The end of the `talk` session is marked as follows:

```
[Connection closing. Exiting]
```

Copying Files to Another Host 12

This chapter explains how to use operating system commands to perform the following tasks:

- Copy files between a local and a remote host
- Copy entire directories (including subdirectories) of files between a local and a remote host
- Copy files between two remote hosts

To determine the host name or online status of a remote host before copying files, use the `finger`, `who`, `rwho`, `ping`, or `ruptime` commands described in Chapter 10.

In addition to the information in this chapter, Chapter 14 provides information on using the UNIX-to-UNIX Copy Program (UUCP) to copy files to and from remote systems.

Note

The security features on the remote host determine whether or not you can copy a file. See your system administrator if you cannot copy a file.

12.1 Copying Files Between a Local and a Remote Host

You can use the following commands to copy files between a local and a remote host:

- `rcp`, described in Section 12.1.1; see Section 12.2 to copy entire directories of files. A host running the Digital UNIX operating system can use `rcp` with a host running any other UNIX based operating system.
- `ftp`, described in Section 12.1.2. You can use `ftp` to copy files between hosts using the following operating systems that also support `ftp`: Digital UNIX and VMS (with UCX).
- `mailx`, described in Section 12.1.3
- `write`, described in Section 12.1.4

12.1.1 Using rcp to Copy Files Between Local and Remote Hosts

When using `rcp` to copy files from a local to a remote host or from a remote to a local host, name the file to be copied first, followed by the destination file, as shown in this `rcp` syntax statement:

```
rcp [ option... ] localfile hostname:file
```

The *localfile* variable identifies the local file you want to copy. The *hostname:file* variable identifies the remote host (*hostname*) followed by a colon (`:`) and the name of the file (*file*) to which the local file is copied.

The following example uses `rcp` to copy the local file, `YTD_sum` from the directory `/usr/reports` on the local host to the file `year-end` in the directory `/usr/acct` on the remote host `moon`:

```
% rcp /usr/reports/YTD_sum moon:/usr/acct/year-end
```

You can also send a file on the local host to a user at a remote host. The following example shows how to copy the file `YTD_sum` from the directory `/usr/reports` on the local host to the file `acct_summaries` in the home directory of user `jones` on the remote host `moon`:

```
% rcp /usr/reports/YTD_sum jones@moon:acct_summaries
```

As used in the preceding examples, the `rcp` command assigns a new creation date and time to the file created from the original. It also assigns file “read-write-execute” permissions according to the host or user directory containing the newly created file.

You may need to preserve the original creation date and access permission mode of the copied file in the new file. As shown in the following example, the `-p` option enables you to preserve the original creation date and time and file access permission of `YTD_sum` in the file, `year-end`:

```
% rcp -p /usr/reports/YTD_sum moon:/usr/acct/year-end
```

If the `-p` option was not entered, a new date and time would have been assigned, and the file access permission would be set to the default assigned by the system administrator for remote host, `moon`.

In the next example, the `-p` option preserves the same file creation and access permissions in the file `acct_summaries` as in the original file, `YTD_sum`:

```
% rcp -p /usr/reports/YTD_sum jones@moon:acct_summaries
```

If the `-p` option was not entered, a new date and time would have been assigned, but unlike the previous example, the file access permission would be the default set by user `jones` through the `umask` command (if any) in the `.login` file. If the `umask` is not set in the `.login` file, the default for remote host `moon` determines the file access permission mode. See `umask(2)` for more information about setting `umask`.

To copy a file from a remote host to a local host, follow the `rcp` syntax statement shown below. The command syntax is the same as copying a local file to a remote host with the exception that `localfile` is the destination file, so it is placed last on the command line:

```
rcp [ option... ] hostname:file localfile
```

12.1.2 Using ftp to Copy Files Between Local and Remote Hosts

The `ftp` command is the interface to the File Transfer Protocol (FTP) and has an extensive set of subcommands (described in Table 12-1, Table 12-2, and Table 12-3) that support the main task of copying files. You can use the `ftp` command to copy files between any two hosts that use the following operating systems: Digital UNIX, UNIX, and VMS (with UCX).

See the `ftp(1)` reference page for a description of the `ftp` command options, which are used primarily for network administration tasks.

Copying files through FTP consists of the following steps:

1. Establishing a session on the remote host
2. Copying the files
3. Disconnecting the session

The `ftp` command has the following syntax:

```
ftp host_name
```

The `host_name` variable specifies the name of the host you want to reach. If you do not specify a `host_name` on the command line, you must use the `ftp` subcommand, `open` (described in Table 12-1) to connect with a remote host.

After you type `ftp`, the `ftp>` prompt is displayed and you are logged in to the remote host. You can then use `ftp` subcommands to perform the following tasks:

- Copy files (See Table 12-1)
- List the contents of a remote directory (See Table 12-2)
- Change the current directory on the remote host (See Table 12-2)
- Append a local file to a remote file (See Table 12-1)
- Copy multiple files (See Table 12-1)
- Escape to the local shell to run commands (See Table 12-3)
- Delete files on remote hosts (See Table 12-2)

Example 12-1 shows how user `alice` on local host `earth` logs on to remote host `moon`, and uses `ftp` subcommands to check the current working directory, list its contents, copy a binary file, and end the session.

Example 12-1: Using `ftp` to Copy a File

```
earth% ftp moon          1
Connected to moon.
220 moon FTLP Server systemname ready.      2
Name(moon:alice): Return                      3
Password: 4
230 User alice logged in                      5
ftp> binary 6
200 Type set to I
ftp> pwd 7
257 "u/alice" is current directory
ftp> ls -l 8
200 PORT command successful.
150 Opening data connection for /bin/ls (192&.9&.200&.1,1026)
(0 bytes)
total 2
-rw-r--r--  1 alice  system    101 Jun  5 10:03 file1
-rw-r--r--  1 alice  system    171 Jun  5 10:03 file2
-rw-r--r--  1 alice  system   1201 Jun  5 10:03 sales
ftp> get sales newsales 9
200 PORT command successful.
150 Opening data connection for testfile (192&.9&.200&.1,1029)
(1201 bytes)
226 Transfer complete.
local:tmp.testfile remote:testfile
ftp> quit 10
221 Goodbye.
earth%
```

- 1 User *alice* enters the `ftp` command at the prompt of local host, *earth* to begin an `ftp` session with remote host, *moon*.
- 2 A message verifying the connection is displayed on the local host.
- 3 User *alice* presses Return at the prompt because her login name is the same on the remote host.
- 4 At the `Password:` prompt, user *alice* enters a valid password that is not displayed.
- 5 The login to the remote host is verified and the `ftp>` prompt appears, establishing the `ftp` session with the remote host.
- 6 User *alice* enters the `binary` subcommand at the `ftp>` prompt to set the file transfer type to binary and FTP verifies it with the message `200 Type set to I.`
- 7 User *alice* enters the `pwd` subcommand to identify the current working directory, and FTP verifies it with the message `u/alice is current directory.`
- 8 User *alice* enters the `ls -l` subcommand to list the contents of the current working directory, `file1`, `file2`, and `sales`.
- 9 User *alice* copies the file `sales` from the remote host to a file called `newsales` on the local host through the `get` subcommand.
- 10 User *alice* enters the `quit` subcommand to end the `ftp` session and returns to the local system prompt.

Table 12-1 describes the `ftp` subcommands that copy files and exit `ftp`. The `binary`, `get`, and `quit` subcommands were used in Example 12-1.

Table 12-1: ftp Subcommands for Connecting to a Host and Copying Files

Subcommand	Description
<code>account [password]</code>	Sends a supplemental password that a remote host other than a Digital UNIX host may require before granting access to its resources. If the <i>password</i> is not specified, the user is prompted for it. The password does not appear on the screen.
<code>ascii</code>	Sets the file transfer type to network ASCII, which is the default. For example, a PostScript file is an ASCII file.
<code>binary</code>	Sets the file transfer type to binary image. This may be more efficient when copying non-ASCII files. For example, a DECwrite file is non-ASCII.

Table 12-1: (continued)

Subcommand	Description
<code>bye</code>	Ends the file copying session and exits FTP; same as <code>quit</code> .
<code>get remfile locfile</code>	Copies the remote file, <i>remfile</i> to the file, <i>locfile</i> on the local host. If <i>locfile</i> is not specified, the remote file name is used locally. See also the <code>runique</code> subcommand.
<code>mget remfile [locfile]</code>	Copies one or more specified files (<i>remfile</i>) from the remote host to <i>locfile</i> in the current directory on the local host (supports wildcard or pattern-matching metacharacter expansion).
<code>mput locfile [remfile]</code>	Copies one or more specified files (<i>locfile</i>) from the local host to <i>remfile</i> on the remote host (supports wildcard or pattern-matching metacharacter expansion).
<code>open host [port]</code>	Establishes a connection with the specified <i>host</i> , if you did not specify it on the command line. If <i>port</i> is specified, FTP attempts to connect to a server at that port. If the <code>autologin</code> feature is set (the default), FTP tries to log the user in to the remote host.
<code>put locfile [remfile]</code>	Stores a file, <i>locfile</i> on the local host, in the file <i>remfile</i> on the remote host. If you do not specify <i>remfile</i> , FTP uses the local file name to name the remote file. See also the <code>sunique</code> subcommand.
<code>quit</code>	Ends the file copying session and exits FTP; same as <code>bye</code> .
<code>recv remfile [locfile]</code>	Copies the remote host file, <i>remfile</i> to the file, <i>locfile</i> on the local host; <code>recv</code> works like <code>get</code> .
<code>runique</code>	Toggles, creating unique file names for local destination files during <code>get</code> operations. If the unique local file name feature is off (the default), FTP overwrites local files. Otherwise, if a local file has the same name as one specified for a local destination file, FTP appends a <code>.1</code> extension to the specified name of the local destination file. If a local file already has the new name, FTP appends a <code>.2</code> extension to the specified name, and so on up to a value of 99. If FTP still cannot find a unique name, it reports an error and the file is not copied. Note that <code>runique</code> does not affect local file names generated from a shell command.

Table 12-1: (continued)

Subcommand	Description
<code>send <i>locfile</i></code> <code>[<i>remfile</i>]</code>	Stores a local file, <i>locfile</i> in the file, <i>remfile</i> on the remote host; <code>send</code> works like <code>put</code> .
<code>sunique</code>	Toggles, creating unique file names for remote destination files during <code>put</code> operations. If the unique remote file name feature is off (the default), FTP overwrites remote files. Otherwise, if a remote file has the same name as specified for a remote destination file, the remote <code>FTLPserver</code> modifies the name of the remote destination file in the same way that <code>runique</code> does, and it must be supported on the remote host.

Table 12-2 describes the `ftp` subcommands that enable you to verify, change, or create the current directory and list its contents before you copy files, if necessary. The `pwd` and `ls` subcommands were used in Example 12-1.

Table 12-2: ftp Subcommands for Directory and File Modification

Subcommand	Description
<code>cd <i>remotedir</i></code>	Changes the working directory on the remote host to <i>remotedir</i> .
<code>cdup</code>	Changes the working directory on the remote host to the parent of the current directory.
<code>delete <i>remfile</i></code>	Deletes the specified remote file.
<code>dir [<i>remdir</i>]</code> <code>[<i>locfile</i>]</code>	Lists the contents of remote directory <i>remdir</i> to the file, <i>locfile</i> on the local host.
<code>lcd [<i>directory</i>]</code>	Changes the working directory on the local host. If you do not specify a <i>directory</i> , FTP uses your home directory.
<code>ls [<i>remdir</i>]</code> <code>[<i>locfile</i>]</code>	Writes an abbreviated file listing of a remote directory, <i>remdir</i> to a local host file, <i>locfile</i> .
<code>mkdir [<i>remdir</i>]</code>	Creates specified directory on remote host.
<code>pwd</code>	Displays the name of the current directory on the remote host.
<code>rename <i>from to</i></code>	Renames a file on the remote host.

Table 12-2: (continued)

Subcommand	Description
<code>rmdir <i>remdir</i></code>	Removes the remote directory <i>remdir</i> from the remote host.

Table 12-3 describes the `ftp` subcommands that provide help or status information directly or by invoking the shell from within `ftp`.

Table 12-3: ftp Subcommands for Help and Status Information

Subcommand	Description
<code>!<i>command</i> [<i>option</i>]</code>	Invokes an interactive shell on the local host.
<code>?[<i>subcommand</i>]</code>	Displays a help message describing the <i>subcommand</i> . If you do not specify <i>subcommand</i> , FTP displays a list of known subcommands. See also the <code>help</code> subcommand.
<code>help [<i>subcommand</i>]</code>	Displays help information. See also the <code>?</code> subcommand.
<code>status</code>	Displays current status of <code>ftp</code> , including the current transfer mode (ASCII or binary), connection status, time-out value, and so on.
<code>verbose</code>	Toggles verbose mode. When verbose mode is on (the default), FTP displays all responses from the remote <code>FTLPserver</code> . Also, FTP displays statistics on all completed file transfers.

The `tftp` command, which is the interface to the Trivial File Transfer Protocol (TFTP), provides another way of copying files. Unlike `ftp`, it does not provide subcommands for any other tasks and is recommended only for tasks performed by the superuser or the installer of the operating system (e.g., copying the operating system kernel). Limited file access privileges are given to the remote `tftp` server daemon, `tftpd`. See the `tftp(1)` reference page for more information.

12.1.3 Using mailx to Copy ASCII Files Between Local and Remote Hosts

The `mailx` command copies ASCII files to a local or remote host, although `mailx` is most often used to send and receive mail messages as described in Chapter 11. You can copy an ASCII file to one or more users through `mailx` by using the left-angle bracket redirection symbol (`<`) as shown in the following syntax:

```
mailx [ option... ] recipient... < filename
```

The *recipient* variable specifies one or more user names or a `mailx` alias to whom you want to send the file, *filename*.

For example, to send the file `schedule` to several users, you could use the `mailx` command, as shown with its `-s` option that indicates the subject of the message:

```
% mailx -s "games" tom jeff craig jim ken < schedule
```

If you create a mail alias of `wombats` (See Section 11.6.1) for these five members of a team called Wombats, you can send the file to that alias, as shown:

```
% mailx -s "games" wombats < schedule
```

12.1.4 Using write to Copy Files Between Local and Remote Hosts

The `write` command copies files to a local or remote host, although `write` is most often used to write messages to other users as tasks described in Section 11.8. After you type `write`, enter the user name of the recipient, a left-angle bracket redirection symbol (`<`), and the name of the file you want to send. For example, to send a file named `letter` in your current directory to user `maria`, enter the following command at the host prompt:

```
% write maria < letter
```

12.2 Copying Directories of Files Between a Local and a Remote Host

The `-r` option of the `rcp` command enables you to copy entire directories of files **recursively** (that is, including files within any subdirectories) between a local and a remote host.

To copy a directory recursively from your local host to a remote host, use the following syntax:

```
rcp -r localdirectory hostname:directory
```

The *localdirectory* variable identifies the local directory that you want to copy recursively. The *hostname:directory* variable identifies the remote host (*hostname*) followed by a colon (:) and the name of the remote directory (*directory*) to which the local directory is copied.

The following example uses `r`cp -r to copy recursively the directory `/usr/reports` from the local host to the directory `/user/status/newdata` on remote host moon:

```
% rcp -r /usr/reports moon:/user/status/newdata
```

You can also copy recursively a directory on your local host to a user at a remote host. The following example shows how to copy the directory `/usr/reports` on the local host to the directory `/user/status/newdata` in the home directory of user `smith` on the remote host moon. This example also uses the `-p` option, as explained in Section 12.1.1, to preserve the original creation date and access permission mode of the directory that is copied in the new directory:

```
% rcp -p -r /usr/reports smith@moon:/user/status/newdata
```

To copy a directory recursively from a remote host to your local host, follow the `r`cp syntax statement shown below. The command syntax is the same as copying a directory recursively from a local to a remote host with the exception that *localdirectory* is the destination file, so it is placed last on the command line:

```
rcp -r hostname:directory localdirectory
```

12.3 Copying Files Between Two Remote Hosts

From your local host, `r`cp can copy a file on one remote host to a file on another remote host. To do this, use the following `r`cp syntax:

```
rcp remhost1:filesend remhost2:file-recv
```

The *remhost1* variable identifies the remote host containing the file you want to send, followed by a colon (:) and the file, *filesend* that you want to send. The last part of the statement identifies the second remote host, *remhost2*, and the file name, *file-recv*, to which the file from *remhost1* will be copied. If only a directory name is given in *file-recv* (as in the example below), *filesend* will be copied there with the same file name.

The following example uses `rcp` to copy the file `spark` from the directory `/u/cave/fred` on remote host `flint` to the directory `/u/hut/barney` on remote host `stone`:

```
% rcp flint:/u/cave/fred/spark stone:/u/hut/barney
```


Working on a Remote Host 13

The chapter explains how to use commands which enable you to:

- Log in to a remote host from your local terminal
- Execute a specified command at a remote host, or if no command is specified, logs in to the remote host
- Log in to a remote host using the Telnet protocol. If `rlogin` is not supported, use `telnet` as an alternative.

Note

Any remote login is subject to the security features on the remote host. If you have difficulty logging in to a remote host, see your system administrator.

Before using any of these commands you might need to know the correct host name or whether a remote host is currently reachable. Use the `finger`, `who`, `rwho`, `ruptime`, and `ping` commands, described in Chapter 10 to find this information.

13.1 Using `rlogin` to Log in to a Remote Host

You can log in to a remote host with `rlogin`, using the following command syntax:

```
rlogin [-l user ] host_name
```

The `-l` option enables you to specify a remote username other than your local username. The `host_name` variable specifies the remote host.

The following steps show how to log in to a remote host `boston` where the login name is the same as that on the local host:

1. Enter the following `rlogin` command followed by the name of the remote host. For example:

```
% rlogin boston
Password:
```

2. Enter your password.

When the system prompt is displayed, you are logged in to the remote host and can enter any command.

3. To close the connection and return to your local host, press Ctrl/d.

If you have an account on a remote host where your login name is different from that on the local host, you must use the `-l` option to log in to the remote host, as shown in the following steps.

1. Enter the `rlogin -l` command followed by the remote login name and the name of the remote host. For example:

```
% rlogin -l celtic boston
Password:
```

2. Enter the password corresponding to the login name, `celtic`.

When the system prompt is displayed, you are logged in to the remote host and can enter any command.

3. To close the connection and return to your local host, press Ctrl/d.

In the following situations, `rlogin` will not prompt for a password:

- If your local host is listed in the `/etc/hosts.equiv` file on the remote host
- If the name of your host (and optionally, your username) is listed in the `.rhosts` file in your home directory on the remote host

For more information on `rlogin`, see the `rlogin(1)` reference page.

13.2 Using `rsh` to Run Commands on a Remote Host

The `rsh` command enables you to run a single command on a remote UNIX based host without logging in there. If you need to run several commands successively, you must log in to the remote host using either `rlogin` or `telnet`.

The `rsh` command has the following syntax:

```
rsh [-l user] host_name command
```

The `-l` option enables you to log in to a remote host where your login name, `user`, is different from that on the local host. If you do not specify the `-l`

option, `rsh` assumes that your login name is the same on both the local and remote hosts. The `host_name` variable specifies the name of the remote host. The `command` variable specifies the command you want to run.

Note

If you do not specify a command to run remotely, `rsh` prompts you for login information to the remote host.

To use `rsh`, one of the following must be true:

- Your local host is listed in the `/etc/hosts.equiv` file on the remote host.
- Your host is listed in the `.rhosts` file in your home directory on the remote host.

In the following example, `rsh` appends a file located on a remote host to a file on the local host. The remote file, `remfile2`, on host `remhost2` is appended to a local file called `locfile`:

```
% rsh remhost2 cat remfile2 >> locfile
```

13.3 Using telnet to Log Into a Remote Host

You can log into a remote host by using the `telnet` command, which implements the Telnet protocol.

Using `telnet` you can:

- Log in to a remote host
- Execute any operating system command on the remote host
- Enter `telnet` subcommands (see Table 13-1) for managing the remote session

The `telnet` command has the following syntax:

```
telnet [ host_name [ port ] ]
```

The `host_name` variable specifies the remote host. If you omit the host name, you can use the `open` subcommand to create a connection after you activate the Telnet utility.

If you do not specify a `port`, the Telnet protocol attempts to contact a Telnet server at the default port.

The following steps show how to use the `telnet` command to log in to a remote host named `host3`, and to use the `telnet` subcommand `status`:

1. Enter the `telnet` command and specify the `host_name` as `host3` (the default port is used):

```
% telnet host3
Trying...
Connected to host3
Escape character is '^]'.
host3 TCLPTelnet service.
login:
```

2. Enter your login name; enter your password.
3. Press `Ctrl/]` (the default escape sequence) to access the `telnet` subcommand prompt, `telnet>`.
4. Enter the `status` subcommand at the prompt, to display status information similar to the following:

```
telnet> status
Connected to host3.
Operating in character-at-a-time mode.
Escape character is '^]'.

```

5. To use another subcommand, press `Ctrl/]` to redisplay the `telnet>` prompt. To display the remote host prompt, press the Return key.
6. To quit the Telnet session from the host prompt, press `Ctrl/d`.
To quit the Telnet session from the `telnet>` subcommand prompt, enter `q`, or press `Ctrl/d`.

You can enter the `telnet` command without any arguments to access the `telnet` subcommand mode, indicated by the `telnet>` prompt.

The `telnet` subcommands are described in Table 13-1. Before entering a subcommand, you must enter the escape sequence, `Ctrl/]`. This sequence notifies the `telnet` program that the following information is not text; otherwise, `telnet` would interpret subcommands as text.

For each subcommand, you only need to type enough letters to uniquely identify the command. For example, `q` is sufficient for the `quit` command. For a complete list of `telnet` subcommands, see the `telnet(1)` reference page.

Table 13-1: telnet Subcommands

Subcommand	Description
? [<i>subcommand</i>]	Displays help information. If a subcommand is specified, information about that subcommand is displayed.

Table 13-1: (continued)

Subcommand	Description
<code>close</code>	Closes the connection and returns to <code>telnet</code> command mode.
<code>display [argument]</code>	Displays all of the set and toggle values if no argument is specified; otherwise, lists only those values that match argument.
<code>open host [port]</code>	Opens a connection to the specified host. The host specification can be either a host name or an Internet address in dotted decimal form. If no port is given, <code>telnet</code> attempts to contact a <code>telnet</code> server at the default port.
<code>quit</code>	Closes a connection and exits the <code>telnet</code> program. Pressing a <code>Ctrl/d</code> in command mode also closes the connection and exits.
<code>status</code>	Shows the status of <code>telnet</code> , including the current mode and the currently connected remote host.
<code>z</code>	Opens a shell on the local host as specified by the <code>SHELL</code> environment variable. When you exit the shell by pressing <code>Ctrl/d</code> , <code>telnet</code> returns to the remote session.

The UUCP Networking Commands **14**

This chapter describes the UNIX-to-UNIX Copy Program (UUCP). Using UUCP enables you to:

- Perform tasks on a remote host
- Transfer files between a local and remote host
- Work in background mode
- Switch back and forth between the local and remote host, performing tasks on either or both, concurrently

For an overview of UUCP system management and tasks, see the *Network and Communications Overview*.

With UUCP, you can connect over a hardwired asynchronous line, a network, or a telephone line (using modems at both ends) to:

- Another workstation
- Another computer running a UNIX based operating system
- A computer running an operating system that is not UNIX based (this requires certain hardware and software)

14.1 UUCP Pathname Conventions

UUCP pathnames follow the same conventions as the operating system with the following exceptions:

- Relative pathnames may not work with all UUCP commands. If not, reenter the command and use the full pathname.
- On hosts that support UUCP, the `/usr/spool/uucppublic` directory is set up for transferring data among other hosts. The brief form of this directory is `~uucp`.
- The pathname for a file on a remote host requires an exclamation point, (!) after the host name. For example, `sea!/research/new` specifies the file `new` in the directory `/research` on host `sea`.
- Sometimes to specify a file, you must provide the names of the remote hosts along the network path. To do so in the Bourne or Korn shells, put an exclamation point (!) after each host name. For example,

`gem!car!sea!/research/cells` specifies the file `cells` in directory `/research` on host `sea`, which is reachable through host `car`; `car` is reachable through host `gem`. In the C shell, the exclamation point (!) will be mistranslated unless you precede it with a backslash (\), for example:

```
gem\
```

14.2 Finding Hosts that Support UUCP

To communicate with a remote host by using UUCP commands from your local host, you must determine which other hosts support the UUCP protocol. The UUCP `uuname` utility displays a list of all hosts with which you can communicate using UUCP from your local host. The following example shows the `uuname` command with output.

```
% uuname
elvis
fab4
```

This example shows that two remote hosts are accessible to the local host through UUCP. To identify the local host, use the `-l` option to the `uuname` command. For example:

```
% uuname -l
music
```

By using UUCP commands among compatible hosts, a user on host `music` can send to or receive files from hosts `elvis` or `fab4`.

For more information, refer to the `uuname(1)` reference page.

14.3 Connecting to a Remote Host

Before you can use UUCP commands, you must connect your local host to the remote host. There are three commands you can use to connect to a remote host:

- The `cu` and `tip` commands establish a full-duplex connection, giving the appearance of being directly logged in to the remote host. This connection enables the simultaneous transfer of data between the hosts.
- The `ct` command establishes a remote connection by letting you dial an attached terminal and log in via a modem and telephone line.

Note

A remote connection is subject to the security features on the local and remote host. See your system administrator for more information.

14.3.1 Using `cu` to Connect to a Remote Host

The `cu` command and its options enable you to connect to a remote host, log in to it, and perform tasks there from your local host. You can perform tasks on each host by switching back and forth between the two. If both hosts use the operating system, you can enter commands on the remote host from your local host.

14.3.1.1 Using `cu` to Connect by Name to a Remote Host

The following steps show how to use the `cu` command to connect from local host `earth` to remote host `moon`, log in to `moon`, and enter a command there:

1. Enter the following `cu` command at the local system prompt; a message verifies the connection:

```
earth% cu moon  
Connected
```

The login prompt for the remote host will be displayed.

When connecting to some remote hosts, you may need to press the Return key several times before a login prompt is displayed.

2. Log in to host `moon` at the login prompt. The system prompt for host `moon` is displayed.
3. At the system prompt, enter any command that host `moon` supports. For example, to list the contents of the `/usr/geog/crater/earthside` directory, enter the following command at the system prompt:

```
moon% ls /usr/geog/crater/earthside  
copernicus.dat  
tycho.dat  
moon%
```

Note

The preceding example may not work for all `cu` connections. It is used here as a brief, general example. See your system administrator for more information.

After logging in to the remote host, you can switch back and forth between it and the local host because they run concurrently. To return to your local host and enter a command there, type a tilde and an exclamation point (~!) followed directly by the command, or wait for the local host prompt to display and then enter the command. To return to the remote host, press Ctrl/d.

14.3.1.2 Using `cu` to Specify a Directly-Connected Remote Host

To connect to a directly-connected remote host, use the `cu` command with the `-l` option to name the hardwired line that connects the two computers. Most of these communication lines have names that are variations of the standard device name, `tty`.

To use the `cu` command with the `-l` option to connect to a remote host with an unknown name, but which uses hardwired device `tttyd0`, enter the following command:

```
earth% cu -l tttyd0
Connected
```

After the connection is made, you can log in to and execute commands on remote host `moon`. Refer to the steps in Section 14.3.1.1.

To return to the local host, type a tilde and an exclamation point (~!) followed directly by the command or wait for the local system prompt to display before entering the command. To return to the remote host, press Ctrl/d.

Note

If you use the `-l` option, but still enter the name of a remote host, no error message will be generated. Instead, `cu` will try to connect to the first available line for the requested host name, ignoring the specified line. If it makes the connection, it may not be the one you intend.

14.3.1.3 Using `cu` to Connect by Telephone to a Remote Host

You can use `cu` to connect by telephone to a remote host whenever the remote host has not been set up to communicate with the local host through UUCP. To do so, the following conditions must be met:

- Both the local and the remote host are connected to modems.
- You know the telephone number of the remote modem and have a valid login on that host.

The following example shows how to use the `cu` command to connect to a remote host that has a long-distance telephone number of 1-612-555-6789. The `-s` option specifies a transmission rate of 300 baud. Assuming that dialing 9 is necessary for an outside dial tone, enter the following `cu` command at the local system prompt:

```
earth% cu -s300 9=16125556789
Connected
```

After the connection is made, you can log in to and execute commands on the remote host. (Refer to the steps in Section 14.3.1.1).

To return to the local host, type a tilde and an exclamation point (~!) followed directly by the command or wait for the local system prompt to display before entering the command. To return to the remote host, press Ctrl/d.

If you do not use the `-s` option to specify a transmission speed, an appropriate rate is selected by default from data in `/usr/lib/uucp/Devices`.

For added security use the `-n` option, which prompts you for the telephone number. This suppresses the display of the phone number with the `ps` command, which would otherwise display the number with the `cu` command that you enter.

Table 14-1 summarizes the `cu` command options and entries. See the `cu(1)` reference page for more information.

Table 14-1: Options to the `cu` Command

Option	Description
<code>-sspeed</code>	Specifies the rate at which data is transmitted to the remote host. The default rate, set during UUCP installation and based on data in <code>/usr/lib/uucp/Devices</code> , should be sufficient for most of your work. Most modems operate at 300, 1200, or 2400 baud, while most hardwired lines are set to 9600 baud or higher. When transferring a large file, you may want to specify the low 300-baud rate for less interference on the line.
<code>-e</code> <code>-o</code>	Specify <code>-e</code> for even or <code>-o</code> for odd parity for data sent to the remote host.

Table 14-1: (continued)

Option	Description
<code>-h</code>	Specify <code>-h</code> to emulate local echoing, to support calls to other hosts that expect terminals to be set to half-duplex mode.
<code>-d</code>	Specify <code>-d</code> to print diagnostic traces.
<code>-n</code>	Specify <code>-n</code> to have <code>cu</code> prompt for a telephone number (for added security)
<code>-l</code> <i>line</i>	<p>Specifies the name of a device (line) for the communication between two computers. The default is either a hardwired asynchronous line, or a telephone line with an automatic dialer such as a modem. If your site has several communication lines, you may want to specify a particular line for your <code>cu</code> link.</p> <p>Usually, you do not have to specify a line or device; the default established during UUCP installation should be sufficient. If you want to connect to a remote computer but do not know its name, you can enter the <code>cu</code> command with the <code>-l</code> option and a variation of the standard device name <code>tty</code> (for example, <code>-ltty1</code>). Ask your system administrator for the device names at your site.</p>
<code>-t</code>	Dials a terminal that has been set to “auto-answer”, and maps carriage return to carriage return/linefeed.
<i>host_name</i>	Specifies the name of the remote host with which you want to establish a connection.
<i>telno</i>	Specifies the telephone number in a remote connection using a modem.

14.3.1.4 Local cu Commands

While connected by `cu` to a remote host, you can use local `cu` commands to perform the following tasks:

- Go back and forth between the local and remote hosts
- Change directories on the local host
- Copy files between local and remote hosts
- Terminate a remote connection

To return temporarily to the local host to work, type a tilde and an exclamation point (~!) at the remote system prompt; wait for the local system prompt to be displayed in the following form, where *local* is the name of the local host:

```
~[ local ] !
```

Instead of waiting for the local system prompt to display, you can enter the command immediately after typing the ~! that accesses the local host. For example, while connected by *cu* to remote host *moon*, you can enter the following command to return to local host *earth* and use the *cat* command to read the */usr/crew/r2/asimov/AI* file:

```
moon% ~!cat /usr/crew/r2/asimov/AI
```

There are three *cu* local commands for tasks that are performed very often. You enter these commands from the remote host to perform tasks on the local host while you continue working on the remote host. These commands are preceded by a tilde and a percent symbol:

```
~%cd directory_name    changes the directory on the local host
~%take from [to]       copies a file from the remote host to the local host
~%put to [from]        copies a file from the local host to the remote host
```

For example, while connected by *cu* to remote host *moon*, you can change the current directory on local host *earth* from */usr/geog/ocean* to */usr/geog/ocean/pacific* by entering the following command:

```
moon% ~%cd pacific
```

While connected by *cu* to remote host *moon*, you can copy the file, */usr/ETI/clavius* to the file */usr/NASA/decode* on local host *earth* by entering the *~%take* local command at the remote system prompt:

```
moon% ~%take /usr/ETI/clavius /usr/NASA/decode
```

While connected by *cu* to remote host *moon*, you can copy the local file */usr/NASA/jupiter* to the file, */usr/ETI/clavius/ha19* on the remote host by entering the *~%put* local command at the remote system prompt:

```
moon% ~%put /usr/NASA/jupiter /usr/ETI/clavius/ha19
```

Note

Before using the `~%take` and `~%put` commands, verify that the destination directory exists. Unlike the `uucp` command, these `cu` local commands do not create intermediate directories during file transfers.

You can transfer only ASCII files with `~%take` and `~%put`. (For example, a PostScript file is an ASCII file, but a DECwrite file is not.)

14.3.1.5 Using `cu` to Connect a Local Computer to Several Remote Computers

You can enter the `cu` to connect host X to host Y, log on to host Y, and then enter the `cu` command there to connect to host Z. You then have one local host computer, X, and two remote host computers, Y and Z.

You can run an operating system command on host Z after you log in there. Then, from Z, you can run commands on the other hosts as follows:

- To run a command on host X, prefix the command with a single tilde (~)
- To run a command on host Y, prefix the command with two tildes (~~)

Table 14-2 summarizes the most common `cu` local commands. For information about other `cu` local commands, refer to `cu(1)`.

Table 14-2: Local `cu` Commands

Command	Description
<code>~.</code>	<p>Logs you off the remote computer and terminates the remote connection.</p> <p>When connected to the remote host over a telephone line using a modem, this command does not always work. In such cases, press <code>Ctrl/d</code> to log off; then type a tilde and a period (<code>~.</code>) at the prompt and press the Return key to terminate the remote connection.</p>
<code>~!</code>	<p>Returns the session from the remote host to the local host. Type a tilde and an exclamation point (<code>~!</code>) at the prompt and enter any command. To return to the remote host, press <code>Ctrl/d</code>.</p> <p>After establishing the <code>cu</code> connection, you can go back and forth between the two hosts by typing <code>~!</code> (to go from remote to local) and pressing <code>Ctrl/d</code> (to go from local to remote).</p>

Table 14-2: (continued)

Command	Description
<code>~%cd <i>directory_name</i></code>	Changes the current directory on the local host to that specified by the <i>directory_name</i> variable. If no directory name is specified, <code>cu</code> changes it to your home directory.
<code>~%take <i>source</i> [<i>dest</i>]</code>	Copies a file from the remote to the local host. If you do not specify a <i>dest</i> destination file on the local host, the <code>~%take</code> command copies the remote file to the local host and assigns the same file name.
<code>~%put <i>source</i> [<i>dest</i>]</code>	Copies a file from the local to the remote host. If you do not specify a <i>dest</i> destination file on the remote host, the <code>~%put</code> command copies the local file to the remote host and assigns the same file name.
<code>~\$cmd</code>	Execute the <i>cmd</i> command on the local host and sends the output to the remote host for execution by the remote shell.

14.3.2 Using `tip` to Connect to a Remote Host

The `tip` command and its options enable you to connect to a remote host, log in to it, and perform tasks there from your local host. You can do tasks on each by switching back and forth between the two. If both hosts use the operating system, you can enter commands on the remote host from your local host.

14.3.2.1 Using `tip` to Connect by Name to a Remote Host

The following steps show how to use the `tip` command to connect from local host `earth` to remote host `moon`, log in to `moon`, and enter a command there:

1. Enter the following `tip` command at the local system prompt; a message verifies the connection:

```
earth% tip moon
Connected
```

The login prompt for the remote host will be displayed.

When connecting to some remote hosts, you may need to press the Return key several times before a login prompt is displayed.

2. Log in to host moon at the login prompt. The system prompt for host moon is displayed.
3. At the system prompt, enter any command that host moon supports. For example, to list the contents of the `/usr/geog/crater/darkside` directory, enter the following command at the system prompt:

```
moon% ls /usr/geog/crater/darkside
copernicus.dat
tycho.dat
moon%
```

Note

The preceding example may not work for all `tip` connections. It is used here as a brief, general example. See your system administrator if necessary.

After logging in to the remote host, you can switch back and forth between it and the local host because they run concurrently. To return to your local host and enter a command there, type a tilde and an exclamation point (~!) followed directly by the command, or wait for the local host prompt to display and then enter the command. To return to the remote host, press `Ctrl/d`.

14.3.2.2 Using `tip` to Connect by Telephone to a Remote Host

You can use the `tip` command to connect by telephone to a remote host if the following conditions are met:

- Both the local and the remote host are connected to modems.
- You know the telephone number of the remote modem or there is an entry for the remote host in `/etc/remote`.

The following steps show how to use the `tip` command to connect to a remote host that has the local telephone number 555-1234 and specifies a transmission rate of 300 baud:

1. Enter the following `tip` command at the local prompt, `jupiter`; a message verifies the connection:

```
jupiter% tip -300 5551234
Connected
```

2. Press the Return key. When connecting to some remote hosts, you may need to press the Return key several times before the remote host's login prompt is displayed.

3. Log in at the remote host login prompt. The connection to your local host is still open, so you can work on the local or remote host.

Note

If you do not specify a transmission speed, the `tip` command uses a 1200-baud rate by default. When transferring a large file, you may want to specify the low 300-baud rate for less interference on the line.

The following steps show how to use the `tip` command to connect, with a 300-baud transmission rate, to a remote host that has a long-distance telephone number of 1-612-555-9876.

1. Assuming that you must dial 9 for an outside dial tone, enter the following `tip` command at the prompt of local host `earth`; a message verifies the connection:

```
earth% tip -300 9,16125559876
Connected
```

2. Press the Return key. When connecting to some remote hosts, you may need to press the Return key several times before the remote host's login prompt is displayed.
3. Log in at the remote host's login prompt. The connection to your local host is still open, so you can work on the local or remote host.

For information about customizing the `/etc/remote` and `/etc/phones` files, refer to the book, *Network Administration* and the `remote(4)` and `phones(4)` reference pages.

Table 14-3 summarizes the `tip` command options and entries. See the `tip(1)` reference page for more information.

Table 14-3: Options to the tip Command

Option	Description
<i>-baud_rate</i>	Specifies data transmission rate to the remote host. The default rate is 1200 baud. Most modems operate at 300, 1200, or 2400 baud, while most hardwired lines are set to 9600 baud or higher. When transferring a large file, you may want to specify the low 300-baud rate for less interference on the line. The baud rate, set when UUCP is installed and customized for your site, is configured according to the hardware used to establish connections.
<i>-v</i>	Displays any variables as they are read (verbose) from the <code>.tiprc</code> file.
<i>host_name</i>	Specifies the remote host to which you want to connect; the <code>tip</code> command connects over a hardwired line or a telephone line using a modem, depending on how your system communications is set up between the local and remote hosts.
<i>telno</i>	Specifies the telephone number in a remote connection, using a modem. Use this method when the remote host name is not recognized by <code>tip</code> (that is, there is no entry in the <code>/etc/remote</code> file).

14.3.2.3 Local tip Commands

While connected by `tip` to a remote host, you can use local commands to perform the following tasks:

- Go back and forth between the local and the remote host
- Change directory on the local host from the remote host
- Copy files between local and remote hosts
- Terminate a remote connection

To return temporarily to the local host and enter commands there, type a tilde and exclamation point (`~!`) at the remote system prompt. The local system prompt will display in the following form, where *shell* is the name of the local shell and *pm*t is the prompt for the local shell, either `%` for the C shell or `$` for the Bourne or Korn shell:

```
~[ shell ] pm!
```

To return to the remote host, press Ctrl/d at the local system prompt. To terminate the `tip` process, type a tilde and press Ctrl/d (~^D).

You can use the following `tip` commands from the remote host to perform tasks on the local host while you continue working on the remote host. These commands are preceded by a tilde:

```
~c directory_name    change the local directory
~t from [to]        copy a file from the remote host to a file on the local host
~<                    copy a file from the remote host to a file on the local host
~p from [to]        copy a file from the local host to a file on the remote host
~>                    copy a file from the local host to a file on the remote host
```

For example, while connected by `tip` to remote host `moon`, you can change the current directory on local host `earth`, from `/usr/geog/polar` to `/usr/geog/polar/arctic` by entering the following command:

```
moon% ~c arctic
```

While connected by `tip` to remote host `moon`, you can copy the `/usr/darkside/temp/dat` file to the `/usr/NASA/bios/temp` file on local host `earth` by entering the following command:

```
moon% ~t /usr/darkside/temp/dat /usr/NASA/bios/temp
```

While connected by `tip` to remote host `moon`, you can copy the local `/usr/NASA/bios/warn` file to the `/usr/darkside/temp/change` file on the remote host by entering the following command:

```
moon% ~p /usr/NASA/bios/warn /usr/darkside/temp/change
```

Note

You can only transfer ASCII files with the `~t` and `~p` commands. (For example, a PostScript file is an ASCII file, but a DECwrite file is not.)

Neither `~t` nor `~p` checks for file transfer errors; the `uucp` command provides this verification.

14.3.2.4 Using `tip` to Connect a Local Computer to Several Remote Computers

You can enter the `tip` command to connect host X to host Y, log on to host Y, and then enter the `tip` command there (if Y supports `tip`) to connect to host Z. You then have one local host computer, X, and two remote host computers, Y and Z.

You can run an operating system command on host Z (if Z is a Digital UNIX host) after you log in there. Then, from Z, you can run commands on the other hosts as follows:

- To run a command on host X, prefix the command with a single tilde (~)
- To run a command on host Y, prefix the command with two tildes (~~)

Note

A command sequence that begins with a tilde (~) can be interpreted by `tip` only if it is at the beginning of the command line.

Table 14-4 summarizes the most common `tip` local commands. For information about other `tip` local commands, refer to `tip(1)`.

Table 14-4: Local tip Commands

Command	Description
<code>~ Ctrl/d</code>	Logs you off the remote computer and terminates the remote connection. When connected to the remote host over a telephone line using a modem, this does not always work. In such cases, press <code>Ctrl/d</code> to log off; then enter <code>~ Ctrl/d</code> at the prompt and press the Return key to terminate the remote connection.
<code>~.</code>	Logs you off the remote computer and terminates the remote connection. When connected to the remote host over a telephone line using a modem, this does not always work. In such cases, press <code>Ctrl/d</code> to log off; then enter <code>~.</code> at the prompt and press the Return key to terminate the remote connection.
<code>~!</code>	Returns the session from the remote host to a shell on the local host. Type a tilde and an exclamation point (<code>~!</code>) at the prompt to enter any command. To return to the remote host, press <code>Ctrl/d</code> . After establishing the <code>tip</code> connection, you can go back and forth between the two hosts by typing <code>~!</code> (to go from remote to local) and press <code>Ctrl/d</code> (to go from local to remote).

Table 14-4: (continued)

Command	Description
<code>~c <i>directory_name</i></code>	Changes the current directory on the local host to that specified by the <i>directory_name</i> variable. If no directory name is specified, <code>tip</code> changes it to your home directory.
<code>~t <i>source</i> [<i>dest</i>]</code>	Copies a file from the remote to the local host. If you do not specify a <i>dest</i> destination file on the local host, the <code>~t</code> command copies the remote file to the local host and assigns the same file name.
<code>~p <i>source</i> [<i>dest</i>]</code>	Copies a file from the local to the remote host. If you do not specify a <i>dest</i> destination file on the remote host, the <code>~p</code> command copies the local file to the remote host and assigns the same file name.
<code>~<</code>	Copies a file from the remote to the local host; the <code>tip</code> command prompts for the command string that will be used on the remote host to display the remote file, and the name of the local file, for example, <code>cat filename</code> .
<code>~></code>	Copies a file from the local to the remote host; the <code>tip</code> command prompts for the name of the local file and sends the the file to the remote host as if it were standard input. The user should set up a command on the remote host to accept this input before executing the <code>~></code> command. For example, <code>remote% cat > destination-file</code> .

Note

The `tip` program uses system prompts and character sequences that match a system's interrupt sequence to signal the end of file transfers through the `~<` or `~>` command. These values are configured in the `/etc/remote` file. See the `remote(4)` reference page for more information.

14.3.3 Using `ct` to Connect to a Remote Terminal with a Modem

The `ct` command enables a user on a remote ASCII terminal with a modem to communicate with a local host with a modem over a telephone line. The remote terminal user can then log in and work on the local host. If there are no available telephone lines, the `ct` command displays a message and asks if you want to wait for one.

The `ct` command is useful in the following situations:

- When secure communications are necessary:
Because the local host contacts the remote terminal, the remote user does not need to know the telephone number of the local host. The local user entering `ct` can monitor the work of the remote user.
- When the cost for the telephone connection should be charged either to the local site or to a specific account on the remote terminal (like a collect call).

The `-h` option can be omitted to emulate making a collect call. The user on a remote terminal enters the `ct` command without the `-h` option.

The following `ct` features are useful under certain circumstances:

- You can instruct `ct` to continue dialing a number until the connection is established or until a set length of time has elapsed.
- You can specify more than one telephone number, and `ct` can dial each modem until a connection is established.

Note

Usually, a user on the remote terminal calls the user on the local host to request a `ct` session. If such connections occur often, your system manager may want to set up UUCP so that a local host automatically enters `ct` to one or more specified terminals at a designated time. For information about customizing UUCP, see your system administrator.

For example, to connect to a remote terminal modem at the same site as yours, enter the following command. The remote modem has a telephone number of 7-6092:

```
earth% ct 76092
Allocated dialer at 1200 baud
Confirm hang_up? (y to hang_up)
```

After entering the command, a message verifies the connection and prompts you to either hang up any other phone lines currently in use or cancel the command.

The following example shows how to use the `ct` command to connect to a remote terminal modem with a local telephone number of 555-0043, specifying 9 for an outside line and the `-w` option for a 2-minute wait for the modem line:

```
earth% ct -w2 9=5550043
Allocated dialer at 1200 baud
Confirm hang_up? (y to hang_up)
```

As before, you are prompted to either hang up any other phone lines currently in use or cancel the command.

The following example shows how to use the `ct` command to connect from local host `earth`, to a remote terminal modem with a long-distance number of 1-201-555-7824, specifying 9 for an outside line and the `-w` option for a 5-minute wait for the modem line:

```
earth% ct -w5 9=12015557824
Allocated dialer at 1200 baud
Confirm hang_up? (y to hang_up)
```

You are prompted to either hang up any other phone lines currently in use or cancel the command.

See the `ct(1)` reference page for more information.

Table 14-5 summarizes `ct` command options and required entries.

Table 14-5: Options to the `ct` Command

Option	Description
<code>-wminutes</code>	Specifies the maximum length of time in minutes that the <code>ct</code> command waits for a line. The <code>ct</code> command dials the remote modem at one-minute intervals until the connection is established or the specified number of minutes has passed. Using the <code>-w</code> option suppresses the messages that <code>ct</code> normally displays if it cannot make the connection.
<code>-xnumber</code>	Produces detailed information about the command's execution on standard error output on the local host, to be used for debugging. The debugging level, <i>number</i> , is a single digit in the range from 0-9; the recommended default is 9.
<code>-v</code>	Enables the <code>ct</code> command to send a running narrative to standard error output.
<code>-h</code>	Prevents <code>ct</code> from breaking the current connection.
<code>-speed</code>	Specifies the data transmission rate of <code>ct</code> ; the default is 1200-baud. Set the baud rate to the baud rate of the terminal to which you are connecting.

Table 14-5: (continued)

Option	Description
<i>telno</i>	<p>Specifies the phone number of the remote modem. You can enter a local or a long-distance number, and specify secondary dial tones such as 9 for an outside line, or an access code.</p> <p>Use an equal sign (=) following a secondary dial tone (9=), and a hyphen (-) for delays, as in 555-5092. Telephone numbers may contain up to 31 characters, including digits 0-9, and any of the following: a hyphen or dash (-), an equal sign (=), an asterisk (*), and a number or pound sign (#).</p>

14.4 Using uux to Run Commands on Remote Hosts

The `uux` command enables you to run commands on a remote host while you work on your local host. If the command does not exist on the remote host, `uux` does not execute, and the remote host will notify you by mail. If the command executes and produces output (for example, `cat` or `diff`), you can specify that `uux` place that output in a file on any specified host.

Note

For security reasons, certain sites may restrict the use of some commands through `uux`. Also, enhanced security features on the local host may affect whether certain commands can be run on remote hosts through `uux`. See your system administrator for more information.

The `uux` command syntax depends on how the command interpreter of a given shell treats special characters. The syntax is the same for the Bourne and Korn shells, but different for the C shell.

Regardless of the shell from which you use `uux`, there are two ways to specify the destination:

```
uux [ option... ] "commandstring > destination"
```

```
uux [ option... ] commandstring \{ destination \ }
```

In the first syntax statement, the right-angle bracket (the redirection symbol) (>) directs the output of the remote command to a destination directory or file. A pair of double quotation marks (" ") encloses the entire command because the redirection symbol, the right-angle bracket (>), is a special character. Whenever you use any of the following characters in a command

line, you must enclose that character or the entire command in double quotation (" ") marks:

- left-angle bracket (<)
- right-angle bracket (>)
- semi-colon (;)
- vertical bar or pipe (|)
- plus sign (+)
- left-bracket ([)
- right-bracket (])
- question-mark (?)

In the second syntax statement, enclose the destination name within braces ({ }). You must type a backslash (\) before each brace because braces are special characters to the shell command interpreter. Without backslashes, the braces would be misinterpreted.

When specifying the pathname of a destination file, you can use a full name, or a pathname preceded by *~user/* where *user* is the name of the user's login directory.

Output files must be writable. If you are uncertain about the permission status of a specific target output file, direct the results of the command to the */usr/spool/uucppublic* directory; *~uucp* is a brief way of specifying this directory from a shell.

14.4.1 Using uux from the Bourne or Korn Shells

The following example shows how the *uux* command uses the operating system *cat* command to concatenate the */u/doc/F1* file located on host *gem*, with the */usr/doc/F2* file located on host *sky*. The result is placed in the */u/doc/F3* file on host *gem*.

```
uux "gem!cat gem!/u/doc/F1 sky!/usr/doc/F2 > gem!/u/doc/F3"
```

In the following example, the task is the same as in the previous command, but braces ({ }) are used instead of the redirection symbol to specify the destination in the *uux* command line. The task is the same as in the previous command, but the destination output is implicit:

```
uux gem!cat gem!/u/doc/F1 sky!/usr/doc/F2 \{gem!/u/doc/F3\}
```

14.4.2 Using uux from the C Shell

To perform the same operation as in the previous section, but in the C shell, enter one of the following `uux` commands:

```
uux "gem\!cat gem\!/u/doc/F1 sky\!/usr/doc/F2 > gem\!/u/doc/F3"
```

The following example uses an implicit destination output file:

```
uux gem\!cat gem\!/u/doc/F1 sky\!/usr/doc/F2 \{gem\!/u/doc/F3\}
```

In the two following examples, `uux` uses the `cat` command to send the `acct6` file from remote host `boston`, as output to the `acct6` file in the public directory on your local host:

```
uux "cat boston\!/reports/acct6 > ~uucp/acct6"
```

The following example uses an implicit destination output file:

```
uux cat boston\!/reports/acct6 \{~uucp/acct6\}
```

14.4.3 Other uux Features and Suggestions

The `uux` command assumes your local host is the default, so you do not need to specify it in the command line. For example, to run the `diff` command to compare the `/u/F1` file on host `car` with the `/u/F2` file on host `sea`, and place the result in the `/u/F3` file on the local host, use the following command:

```
uux "diff car\!/u/F1 sea\!/u/F2 > /u/F3"
```

You can also represent the local host by using just an exclamation point, as in the following example:

```
uux "!diff car\!/u/F1 sea\!/u/F2 > !/u/F3"
```

When you specify the pathname source file in commands such as `diff` or `cat`, you can include the following shell pattern-matching characters which the remote host will interpret: the question-mark (`?`), the asterisk (`*`), the left-bracket (`[`), and the right bracket (`]`). Enclose these characters either within two backslashes (`\ ... \`) or within quotation marks (`" ... "`) so that the local shell does not interpret the characters before `uux` sends the command to the remote host. Do not use pattern-matching characters in destination names.

If you use the left-angle bracket (`<`), the right angle-bracket (`>`), the semicolon (`;`), and the pipe (`|`) shell characters, place them within backslashes (`\ ... \`), quotation marks (`" ... "`), or place the entire command line within backslashes or quotation marks.

Note

The shell redirection characters, two left-angle brackets (<<) and two right-angle brackets (>>), do not work in UUCP.

Table 14-6 summarizes `uux` command options and required entries. See the `uux(1)` reference page for more information.

Table 14-6: Options to the `uux` Command

Option	Description
<code>-n</code>	Cancels notification through <code>mailx</code> that usually occurs when a command fails to execute on the designated host. The <code>-n</code> and <code>-z</code> options are mutually exclusive.
<code>-z</code>	Sends a message through <code>mailx</code> when command execution fails on the designated host. The <code>-n</code> and <code>-z</code> options are mutually exclusive.
<code>-j</code>	Displays the job identification number of the <code>uux</code> request that runs the remote command; use this number with the <code>uustat</code> command. See Section 13.8.1 for more information.
<i>commandstring</i>	Specifies any command accepted by the designated host. For more information on the command formats, see Section 13.4.
<i>destination_name</i>	Specifies the host and file for storing the output of the command run on a remote host. For example, if you want to list all the files in a directory on a remote host, you can use <code>uux</code> to place the listing in a file on your own host by entering the appropriate destination name. For more information on the destination formats, see Section 13.4.

14.5 Using UUCP To Send and Receive Files

On UNIX based computers (such as Digital UNIX that support the UUCP protocol), you can use the `uucp` command to copy one or more files from one computer to another. You can use `uucp` to copy files as follows:

- Between local and remote hosts
- Between two remote hosts
- Between two hosts through an intermediate host

- Within your local host

To facilitate file transfers, many sites make the public UUCP directory, `usr/spool/uucppublic`, available. This directory provides read and write access to all users and bypasses security restrictions. The brief way to specify this directory is `~uucp` or `~/` in a `uucp` command.

Note

File transfer through `uucp` is subject to security features on either host. The `uucp` utility does not display error messages for failed file transfers. For more information, see your system administrator.

The system administrator defines security restrictions to prevent unwarranted use by remote users, so only certain directories and files may be accessible for sending or receiving files.

14.5.1 Using UUCP to Copy Files in the Bourne and Korn Shells

From the Bourne or Korn shells, you can specify `uucp` file names without using a backslash (`\`) before the exclamation point (!) that precedes the host name of the destination file. For example, to copy the `star` file from local host `earth` to the `/sun/stats` file in the public directory on the remote host `sky`, enter the following command:

```
earth% uucp star sky!~/sun/stats
```

To copy the same file and explicitly identify the `/usr/spool/uucppublic` directory, enter the following command:

```
earth% uucp star sky!/usr/spool/uucppublic/sun/stats
```

If you need to copy a file to a remote host whose address is unknown to your local host, you can do so via another host that knows the remote host's address. You can copy a local file to a remote host by first sending it to one or more intermediate hosts, separating each host name by an exclamation point (!). For example, to copy the local file `star` to the `/sun/stats` file on the remote host `sky` by first sending it through the intermediate host, `mlkway`, enter the following command:

```
earth% uucp star mlkway!sky!~uucp/sun/stats
```

You can use `uucp` from your local host to copy a file from a remote host to your local host. For example, to copy the `/cells/type1` file from remote host `biochem` to the local file, `/dna/sequence`, enter the following command from local host `earth`:

```
earth% uucp biochem!/cells/type1 /dna/sequence
```

You can copy multiple files from a remote host to a local host by using a pattern-matching character to specify files. For example, to copy all files with names beginning with `report` from the `/geog/survey` directory on remote host `moon` to the `~uucp` public directory on local host `earth`, enter the following command:

```
earth% uucp moon/geog/survey/report* ~uucp
```

14.5.2 Using UUCP to Copy Files in the C Shell

In the C shell, the exclamation point (!) has a special meaning. To prevent the command interpreter from mistranslating it, you must precede it with a backslash (\) in a pathname.

For example, to copy the `/usr/NASA/ctrl-specs` file from local host `earth` to the `~uucp` public directory on remote host `luna7`, enter the following command from the local host:

```
earth% uucp /usr/NASA/ctrl-specs luna7\!~uucp
```

To copy the `plan9` file from the `/usr/reports/exobio` directory on remote host `luna7` to the `~uucp` public directory on local host `earth`, enter the following command:

```
earth% uucp luna7\!/usr/reports/exobio/plan9 ~uucp
```

To copy all files with names beginning with `msg` from the `/sensory/visual/earthrise` directory on the remote host `luna7` to the `~uucp` public directory on local host `earth`, you can enter the following command:

```
earth% uucp luna7\!/sensory/visual/earthrise/msg'*' ~uucp
```

Here, the pattern-matching character, the asterisk (*) in the source file names is enclosed within single quotation marks to prevent misinterpretation.

In the next example, the same files are copied to `~uucp`, but the entire pathname of the source files is enclosed in double quotation marks to prevent misinterpretation:

```
earth% uucp "luna7\!/sensory/visual/earthrise/msg*" ~uucp
```

Table 14-7 summarizes `uucp` command options and required entries. See the `uucp(1)` reference page for more information.

Table 14-7: Options to the UUCP Command

Option	Description
<code>-d</code>	Creates intermediate directories needed when copying a source file to a destination file on a remote host. Entering <code>uucp</code> with the destination pathname creates the required directory. The <code>-d</code> option is the default.
<code>-f</code>	Do not create intermediate directories during the file transfer.
<code>-j</code>	Displays the job identification number of the transfer operation; use with the <code>uustat</code> command to check transfer status, or with <code>uustat -k</code> to terminate the transfer. See Section 13.8.1 for more information.
<code>-m</code>	Specifies that <code>uucp</code> send mail to the requester to verify copying of destination file on a remote host; no mail is sent for a local transfer.
<code>-nusername</code>	Notifies the recipient, <i>username</i> on the remote host, that a file has been sent; no mail is sent for a local transfer.
<i>source_file</i>	Specifies the pathname of the file that you want to send or receive. For more information about UUCP pathnames, see Section 13.1.
<i>destination_name</i>	Specifies the pathname of the file (or directory) that receives the copy. For more information about destination file pathnames, see Section 13.5.1 and Section 13.5.2.

14.6 Using `uuto` with `uupick` to Copy Files

The `uuto` command copies the file you specify to the public directory on the destination host where it is obtained by the recipient through `uupick`. The `rmail` program notifies the recipient when the file arrives.

Note

Any file transfer is subject to the security features on the local and remote hosts. See your system administrator for more information.

For example, to send the `/usr/bin/data/junk` file from local host `moe` to user `curly` on remote host `stooge`, enter the following command:

```
moe% uuto /usr/bin/data/junk stooge!curly
```

The `uuto` command copies the file to the `/usr/spool/uucppublic/receive/curly/moe` file on host `stooge`. Next, the `rmail` utility sends user `curly` a mail message stating that the file has arrived. User `curly` can then enter the `uupick` command to access the file and save, move, or delete it. In the following example, user `curly` enters the `uupick` command on host `stooge`; the response from `uupick` follows:

```
stooge% uupick
from system moe: file junk
?
```

At the `uupick` question mark (?) prompt, user `curly` enters the `d` and `q` options to delete the file and exit from `uupick`:

```
? d
? q
```

Table 14-8 summarizes the `uupick` file handling options, entered at the ? `uupick` prompt.

Table 14-8: Options to the `uupick` Command

Option	Description
*	Displays available <code>uupick</code> file-handling options.
Return	Pressing the Return key signals <code>uupick</code> to get the next file.
a [<i>dir</i>]	Moves all <code>uuto</code> files from the public directory to the specified directory on the local host; specify the directory by using a full or relative pathname. The default is the directory where you enter <code>uupick</code> .
d	The <code>d</code> option deletes the current file obtained by <code>uupick</code> .
m [<i>dir</i>]	Moves a file to a directory specified by either full or relative pathname; the default is the current directory.
p	Displays the file on your workstation.

Table 14-8: (continued)

Option	Description
<code>q</code> or <code>Ctrl/d</code>	The <code>q</code> (quit) option enables you to exit from <code>uupick</code> without displaying, moving, or deleting any file in the public directory. You can also press <code>Ctrl/d</code> to quit.
<code>!command</code>	Enables you to leave <code>uupick</code> and return to the operating system prompt where you can run a command. After the command executes, control returns to <code>uupick</code> .

See the `uupick(1)` reference page for more information.

14.7 Using `uuto` to Send a File Locally

You can also use `uuto` to send a file to another user on your local host. However, the recipient does not receive a mail message indicating the file transfer. For example, user `shemp` can send the file `/usr/bin/data/status` to user `larry` on local host `stooge`, where each is logged in:

```
stooge% uuto /usr/bin/data/status larry
```

Table 14-9 summarizes `uuto` command options and required entries. See the `uuto(1)` reference page for more information.

Table 14-9: Options to the `uuto` Command

Option	Description
<code>-m</code>	Notifies sender when <code>uuto</code> copies a source file to the specified user name and host.
<code>-p</code>	Usually, <code>uuto</code> copies the source file to: <code>/usr/spool/uucppublic/receive</code> <code>/username/host/file</code> . The <code>-p</code> option sends the source file to the spool directory on the local host before transferring a copy of it to the public directory on the specified host.
<code>file_name</code>	The pathname of the source file.

Table 14-9: (continued)

Option	Description
<i>destination_name</i>	The pathname to the location where you want to copy the source file. The <i>destination_name</i> must include the user name of the person receiving the file, and has the form, <i>host!username</i> , where <i>host</i> is the name of the remote computer and <i>username</i> is the user name of the recipient. When copying a file on your local host, the <i>destination_name</i> can be simply the name of the user to whom you are sending the file.

14.8 Displaying Job Status of UUCP Utilities

The UUCP utilities include three commands, `uustat`, `uulog`, and `uumonitor` for displaying status information about UUCP jobs, as described in the following sections.

14.8.1 The `uustat` Command

The `uustat` command supports UUCP jobs by providing the following:

- Status information of file transfers requested by `uucp` and `uuto`
- Status information of command executions requested by `uux`
- Limited control of jobs queued to run on a remote computer
- Cancellation of copy requests from `uucp`

Status reports from `uustat` display on your workstation screen in this basic form; variations depend on the `uustat` option.

```
jobid date/time status system_name username size file
```

Note

Any status display operation is subject to the security features on the local and remote hosts. See your system administrator for more information.

Entering `uustat` without options displays the status information for all the UUCP commands that you have entered since the last time the holding queue was cleaned up.

To report the status of jobs requested by a specific user, use the `-u` option, as shown here, for user hugh:

```
% uustat -u hugh
```

Two types of output information, each produced by a `uustat` option, are the current queue and the holding queue: The output of the `uustat -q` command is the **current queue**, which lists the UUCP jobs either queued to run or currently executing on one or more remote hosts. The output of the `uustat -a` command is the **holding queue**, which lists all jobs that have not executed during a set period of time.

Note

After the set time period has elapsed, delete the entries in the holding queue manually with the `uucleanup` command or automatically through the `uudemon.cleanu` script. The `uudemon.cleanu` script has an entry in `/usr/spool/cron/crontabs/uucp` which is activated by the `/etc/cron` daemon. For more information about cleaning up UUCP queues, see the `uucleanup(8)` reference page or your system administrator.

14.8.1.1 Displaying the Holding Queue Output with a `uustat` Option

To examine the status of all UUCP jobs in the holding queue, enter the `uustat -a` command as shown here with example output:

```
% uustat -a
sunC3113 Thu Jun 04 17:47:25 1996 S sun doc 289 D.car471afd8
gemN3130 Thu Jun 04 09:14:30 1996 R gem geo 338 D.car471bc0a
seaC3120 Wed Jun 03 16:02:33 1996 S sea doc 828 /u/doc/tt
seaC3119 Wed Jun 03 12:32:01 1996 S sea msg rmail doc
```

This example output consists of the following seven fields:

- Field 1 – job ID of the operation; if you need to cancel a process that is still on the local computer, you would use this field as input to the `uustat` command with the `-k` option, for example,

```
% uustat -k seaC3119
```

- Field 2 – date and time that the UUCP command was entered
- Field 3 – S or an R, depending on whether the job sends or receives a file
- Field 4 – name of the hosts where the command was entered
- Field 5 – username of the person who entered the command

- Field 6 – file size or, in the case of remote execution (as in the last output line), the name of the remote command (`rmail`).
- Field 7 – when the size is given in field 6, (as in the first three output lines) the file name is displayed in this field

The file name can be either the name given by the user, such as `/u/doc/tt` or a name that UUCP assigns internally to data files associated with remote executions, such as `D.car471afd8`.

To report the status of all UUCP jobs in the holding queue requested by a specific host, enter the `uustat -s` command as shown here with example output, for host `sky`:

```
% uustat -s sky
skyNlbd7 Wed Jun 03 12:09:30 1996 S sky doc 522 /user/doc/A
skyClbd8 Wed Jun 03 12:10:15 1996 S sky doc 59 D.3b2a12ce4924
skyC3119 Wed Jun 03 12:11:18 1996 S sky doc rmail msg
```

This output is the same as the output produced by the command `uustat -a -s sky`.

14.8.1.2 Displaying the Current Queue Output with `uustat` Options

To examine the status of all UUCP jobs currently executing or queued to run on each host (the current queue) enter the `uustat -q` command as shown here with example output:

```
% uustat -q
sea 3C Mon Jul 13 09:14:35 1996 NO DEVICES AVAILABLE
sun 2C Mon Jul 13 10:02:22 1996 SUCCESSFUL
gem 1C (2) Mon Jul 13 10:12:48 1996 CAN'T ACCESS DEVICE
```

This example output consists of the following five fields:

- Field 1 – host name
- Field 2 – number of files, either command (C), or executable (X), in the holding queue for that host
- Field 3 – number of days (if one or more) that the file has been in the holding queue
- Field 4 – date and time when UUCP last tried to communicate with the host in field 1.
- Field 5 – status message of the interaction

See the `uustat(1)` reference page for more information.

Table 14-10 summarizes `uustat` command options and required entries.

Table 14-10: Options to the uustat Command

Option	Description
-a	Displays information for all jobs in the holding queue, regardless of the user who entered the original UUCP command.
-k <i>jobid</i>	Cancels the UUCP process specified by <i>jobid</i> . You can cancel a job only if you entered the UUCP command specified by <i>jobid</i> . Anyone with superuser privileges can also cancel UUCP requests.
-m	Reports on the status of your most recent attempt to communicate with another computer through UUCP. For example, the status is reported as successful if the UUCP request executed. If the job was not completed, UUCP reports an error message, such as <code>Login failed</code> .
-p	Useful only to someone with superuser privileges, it runs a <code>ps -flp</code> (process status: a full, long list of specified process IDs) for all PID numbers in the lock files.
-q	Lists the jobs currently queued for each computer. These jobs are either waiting to execute, or in the process of executing. If a status file exists for the computer, UUCP reports its date, time, and the status information. Once the process is completed, UUCP removes the job listing from the current queue.
-r <i>jobid</i>	Rejuvenates the UUCP process specified by the job identification number. This option enables you to mark files in the holding queue with the current date and time, to ensure that the cleanup operation does not delete these files until the allotted job modification time ends.
-shost	Reports the status of all UUCP requests that users have entered to run on the computer specified by the host entry.
-username	Reports the status of all UUCP requests entered to run that were specified by the user named in the user entry. You can use both the <code>-shost</code> and the <code>-username</code> options with the <code>uustat</code> command to get a status report on all UUCP requests entered by a specified user and host.

14.8.2 Using the uulog Command to Display UUCP Log Files

Whenever the local host uses the `uucp`, `uuto`, or `uux` commands, UUCP log files are created for each remote host. The `uulog` command displays these log files. Use `uulog` to display a summary of `uucp`, `uuto`, and `uux` command requests by user or by host.

The `uulog` command displays the contents of the log file activity of either of the following daemons:

- The `uucico` daemon, called by `uucp` and `uuto`

The activity of this daemon is logged in
`/usr/spool/uucp/.Log/uucico/host`.

- The `uuxqt` daemon, called by `uux`

The activity of this daemon is logged in
`/usr/spool/uucp/.Log/uuxqt/host`.

To display just the `uuxqt` log file, use the `-x` option of `uulog`, as follows:

```
% uulog -x
```

The `uulog` command also enables you to display the `uucico` log file or the file transfer log for any host, or only a specified number of lines at the end of either log file. For example, to display the `uucico` log file for host `sky`, use the `-s` option as follows:

```
% uulog -s sky
```

To display the last 40 lines of the file transfer log for host `sky`, use the `-f` option and the number option as shown:

```
% uulog -f sky -40
```

Table 14-11 summarizes `uulog` command options and required entries.

Table 14-11: Options to the uulog Command

Option	Description
<code>-fhost</code>	Performs a <code>tail -f</code> on the file transfer log for the specified <i>host</i> , displaying the end of the log file. Use the <code>Interrupt</code> key sequence to leave the file and return to the prompt.
<code>-s[host]</code>	Prints information about copy requests involving the specified host. If no host is specified, information is displayed for all hosts. Host names can contain only ASCII characters.
<code>-x</code>	Displays the <code>uuxqt</code> log file.

Table 14-11: (continued)

Option	Description
<i>-number</i>	Displays the last lines of the log file. The number of lines is determined by <i>number</i> .

14.8.3 Monitoring UUCP Status

The `uumonitor` command is helpful for detecting a host whose status has changed due to a backlog of jobs, a temporary shutdown, or a change of either the phone number or log-in password.

The `uumonitor` output consists of the following six fields:

- Field 1 – host name
- Field 2 – number of command files queued for the remote host; if too large (for example, 100-1000, depending on the host), then the cause of the backlog should be determined
- Field 3 – number of requests for remote execution from the remote host
- Field 4 – result of the most recent attempt to connect to the remote host
- Field 5 – number of remote host login failures, not including failed dial attempts; if greater than 20, no further attempts are made
- Field 6 – time of last status entry

For more information, see the `uumonitor(8)` reference page.

A Beginner's Guide to Using vi **A**

Whether you are writing memos or modifying C programs, editing text files is one of the most common uses of any computer system. The `vi` text editor (hereafter known as `vi`) is particularly well-suited for the day-to-day text editing tasks of most computer users. You can quickly and easily open a file, edit it, and save the results using `vi`.

The `vi` text editor is a full-featured text editor with the following major features:

- Fast processing, especially on startup and global operations
- Full screen editing and scrolling capability
- Separate text entry and edit modes
- Global substitution and complex editing commands using the underlying `ex` commands
- Access to operating system level commands
- Ability to customize system parameters and keyboard mappings

This appendix shows you how to use the basic features of `vi`. After completing the exercises in this appendix, you will be able to:

- Create and save a new file
- Access (open) an existing file
- Move the cursor within the file
- Enter new text
- Change existing text
- Search for strings
- Move and copy text
- Make global substitutions
- Write all or part of the text to a file
- Delete, move, or copy blocks of text
- Customize your editing environment

This appendix only provides an introduction to the features of `vi`. If you want to learn more, see the `vi(1)` reference page. You may also read one of the many books on the market that describe `vi`'s advanced features.

This appendix is divided into three sections. The first section gets you started with `vi`. The second section shows you some advanced techniques for speeding up your work. The third section shows you how to take advantage of the power of the underlying `ex` commands.

A.1 Getting Started

This section shows you how to create a file with `vi`, save the file, move the cursor around in the file, add text, delete text, and modify text.

A.1.1 Creating a File

To create the file, `my.file`, that will be used in the examples throughout this appendix, enter the `vi` command as shown below:

```
$ vi my.file Return
```

Your screen will look like this:

```
~
~
~
~
~
~
~
~
~
~
"my.file" [New file]
```

The lines beginning with tildes (~) represent the blank lines in the file. Because `my.file` is empty, all lines in the file begin with a tilde (~).

The `vi` editor has two modes: **command mode** and **input mode**.

Command mode is the mode `vi` is in when it is started. In command mode, the characters you enter are interpreted as commands for manipulating the text. When `vi` is in input mode, the characters you enter are interpreted as text.

When you create a new file with the `vi` command, the `vi` editor is in the command mode. That is, `vi` is waiting for you to enter a command. However, at this point you want `vi` to be in the input mode so you can insert text into `my.file`, which is empty.

Put `vi` into the input mode by typing:

```
i
```

The `i` command will not be displayed on the screen. The `vi` editor is now in the input mode and `vi` will interpret all characters that you type to be text.

In the sample text below, notice the use of the Escape key on the last line of input and the use of the `:wq` command to save the file and exit the `vi` editor. Type the sample text exactly as shown. If you make a mistake, use the Backspace key to correct it; press the Return key where indicated to move to the next line of text:

```
You can use this text file Return
to experiment with vi. Return
The examples shown here Return
will teach you the basics of vi. Escape
~
~
~
~
~
~
:wq

"my.file" 4 lines, 108 characters
$
```

Note

Depending upon how your terminal or workstation is set up, the Escape key may be programmed to perform a different function. It is possible that one of the function keys on your keyboard (possibly F11) may have been set up to perform the escape function. See your system administrator if your Escape key does not operate properly.

Pressing the Escape key while `vi` is in the input mode puts `vi` back into the command mode; once in the command mode `vi` interprets anything you type to be a command. The `:wq` command writes (saves) the file with the name `my.file` into your current directory and quits the `vi` editor.

The format of the `:wq` command is much different than other `vi` commands because `:wq` is not a `vi` command; it is an `ex` command. When you press a colon (`:`) when `vi` is in the command mode, notice that it appears at the bottom of the screen. The colon (`:`) begins all `ex` commands from within `vi`. All `ex` commands are executed when `vi` is in the command mode. You must press the Return key after the command to signify to `ex` that you are finished entering the command. See Section A.3 to learn more about `ex` commands.

If you lose track of which mode `vi` is in, press the Escape key a few times to make sure `vi` is in the command mode. If your system is so configured, you will hear a bell when you press the Escape key that signals that `vi` is indeed in the command mode.

The Escape key and its use in `vi` and exiting `vi` using several different methods are described in more detail later in this appendix.

The text you just entered in `my.file` will be used in the remaining examples in this appendix.

A.1.2 Opening an Existing File

Whether you are creating a new file or opening an existing file, the syntax for using `vi` is the same:

```
vi filename
```

To open the `my.file` file, enter the `vi` command as follows:

```
$ vi my.file
```

Your screen should look like this:

```
You can use this text file  
to experiment with vi.  
The examples shown here  
will teach you the basics of vi.
```

```
~  
~  
~  
~  
~  
~
```

```
"my.file" 4 lines, 108 characters
```

The text you entered in the file will be displayed at the top of the screen. The lines beginning with tildes (~) represent the blank lines in your file. The text at the bottom of the screen shows the name of the file, the number of lines in the file, and the number of characters in the file.

A.1.3 Saving a File and Quitting vi

In the previous example, you learned that the `:wq` command saved the file and quit the `vi` editor. However, there are several other options available to save and quit a file:

- Save a file and continue working in it

- Save the file and quit (exit) `vi`
- Quit `vi` without saving the changes made to the file

If you are working on a large text file and have been adding, changing, and deleting a lot of information, it is suggested to save the file often (perhaps every 10 minutes) to protect against potential data loss. The `write` command is used to save an entire file to the current directory. The format of the `write` command is:

```
:w filename
```

The entry of *filename* is optional and is used only when you want to save a file under a different filename. Omitting *filename* from the command automatically saves a file to its current file name. When you enter the `:w` command, the current file name, number of lines, and number of characters is displayed at the bottom of your screen. If you entered a new file name, the new file name will be displayed.

Note

If you specify a new file name with the `:w` command, you will have two files saved in your directory: the new file name you just entered and the original file name.

If you are finished making changes to a file, you can save the file and quit `vi` at the same time. The format of the `write` and `quit` command is:

```
:wq
```

The `:wq` command saves a file to the same file name, quits `vi`, and brings you back to your shell prompt.

You also have the option to quit a file and `vi` simultaneously without saving the changes you may have made. This option is useful if, for example, you have deleted many lines of information by mistake and you want to start all over again. Quitting `vi` will restore your file to its original state. However, quitting `vi` to restore a file to its original state will only work if you have not saved the file previously during the current editing session. To quit your file and `vi` without saving your changes enter:

```
:q!
```

Quitting a file with the `:q!` command will not delete the file from your directory. Your file will still reside in the directory, but it will not contain any of the changes you may have made.

Table A-1 summarizes the commands used to save files and quit the `vi` editor.

Table A-1: Write and Quit Command Summary

Command	Result
<code>:w</code>	Saves the entire file to the current file name; does not exit the <code>vi</code> editor.
<code>:w filename</code>	Saves the entire file to the new file name; does not exit the <code>vi</code> editor. The new file name and original file name reside in the directory.
<code>:wq</code>	Saves the entire file to the current file name and exits the <code>vi</code> editor simultaneously.
<code>:q!</code>	Quits the file; exits the <code>vi</code> editor; does not save any changes made to the file since the last time the file was saved.

A.1.4 Moving Within a File

If you have closed `my.file`, reopen it by using the command:

```
$ vi my.file
```

The cursor should be on the first character in the file: the `Y` in the word `You`.

As mentioned previously in this appendix, `vi` is in command mode at start up. In command mode, the characters you enter are treated as commands rather than as text input to the file.

A.1.4.1 Moving the Cursor Up, Down, Left, and Right

Certain keys on the keyboard have been designated to be **movement** keys when `vi` is in the command mode. The following letters on the keyboard control cursor movement:

- `h` (move the cursor one character to the right)
- `j` (move the cursor down one line staying in the same position)
- `k` (move the cursor up one line staying in the same position)
- `l` (move the cursor one character to the left)

Using the movement keys, move the cursor to the first letter of the word `experiment` by typing:

```
111j
```

If your keyboard is equipped with arrow keys, you may be able to use the arrow keys to move left, right, up, or down as well. However, using the `h`, `j`, `k`, and `l` keys allows you to keep your fingers on the main section of the keyboard for faster typing. On some keyboards, the `h`, `j`, `k`, and `l` keys are repetitive keys. That is, holding the key down will repeat the key action until you release the key. For instance, holding down the `j` key will rapidly scroll through the lines in a file.

In the command mode, the Return key acts as a cursor movement key. Pressing the Return key moves the cursor to the first character of the next line. This movement differs from the `j` movement key because the Return key positions the cursor at the first character of the next line whereas the `j` moves the cursor to the same character position on the next line.

In the command mode, the hyphen (`-`) moves the cursor to the first character of the previous line. This feature is useful to move backward through files. This movement differs from the `k` movement key because the hyphen (`-`) positions the cursor at the first character of the previous line whereas the `k` moves the cursor to the same character position on the previous line.

If you tested any of the cursor movement keys described above, make sure your cursor is positioned at the first letter of the word `experiment` before continuing to the next section.

A.1.4.2 Moving the Cursor by Word, Line, Sentence, and Paragraph

You can use the `w` command to move the cursor by whole word boundaries. The `w` command moves the cursor forward to the beginning of the next word. Move the cursor to the beginning of the word `with` by typing:

```
w
```

You can use the `b` command to move backward to the beginning of the previous word. For example, move to the beginning of the word `experiment` by typing:

```
b
```

Now see what happens when you do not use the `b` command from the beginning of a word by typing:

```
1111b
```

The cursor returns to the beginning of the word `experiment`.

The word motion commands will wrap to the next or previous text line when appropriate. Move the cursor to the beginning of the word `text` by typing:

```
bbb
```

Notice how the cursor moved backward and wrapped around to the previous line.

There are a few other interesting movement commands you should know about. The zero (`0`) moves the cursor to the beginning of the current line, and the dollar sign, (`$`) moves the cursor to the end of the current line.

The close parenthesis (`])` moves the cursor to the beginning of the *next sentence*, and the open parenthesis (`[`) moves the cursor to the beginning of the *previous sentence*.

The right brace (`})` moves the cursor to the beginning of the *next paragraph*, and the left brace (`{`) moves the cursor to the beginning of the *previous paragraph*.

A.1.4.3 Moving and Scrolling the Cursor Forward and Backward Through a File

In larger files, you can move the cursor by whole screenfuls by pressing certain control keys:

- `Ctrl/F` moves the cursor one full screen forward
- `Ctrl/B` moves the cursor one full screen backward
- `Ctrl/D` moves the cursor and scrolls down (forward) a half screen
- `Ctrl/U` moves the cursor and scrolls up (backward) a half screen

The following uppercase letters also designate cursor movement over large boundaries of text:

- The `H` command moves the cursor *Home*; that is, to the first character in the file
- The `G` command instructs the cursor to *Go* to the last line in the file

A.1.4.4 Movement Command Summary

The `vi` text editor has many more cursor movement commands. When you have learned the basics documented in this appendix, refer to the `vi(1)` reference page for more information.

Table A-2 summarizes the cursor movement commands. The cursor movement keys are in effect only when `vi` is in the command mode.

Table A-2: Cursor Movement Command Summary

Command	Result
<code>h</code>	Move the cursor one character to the right.
<code>j</code>	Move the cursor down one line in the same position.
<code>k</code>	Move the cursor up one line in the same position.
<code>l</code>	Move the cursor one character to the left.
Return key	Move the cursor to the beginning of the next line.
<code>-</code>	Move the cursor to the beginning of the previous line.
<code>w</code>	Move the cursor forward to the beginning of the next word.
<code>b</code>	Move the cursor backward to the beginning of the previous word.
<code>0</code>	Move the cursor to the beginning of the current line.
<code>\$</code>	Move the cursor to the end of the current line.
<code>)</code>	Move the cursor to the beginning of the next sentence.
<code>(</code>	Move the cursor to the beginning of the previous sentence.
<code>}</code>	Move the cursor to the beginning of the next paragraph.
<code>{</code>	Move the cursor to the beginning of the previous paragraph.
<code>Ctrl/D</code>	Scroll down (forward) a half screen.
<code>Ctrl/F</code>	Move the cursor forward one screen.
<code>Ctrl/B</code>	Move the cursor backward one screen.
<code>Ctrl/U</code>	Scroll up (backward) a half screen.
<code>H</code>	Move the cursor home (to the first character in the file).
<code>G</code>	Move the cursor to the last line of the file.

A.1.5 Entering New Text

To enter new text into a file, `vi` must be in the input mode. In input mode, the characters you enter are inserted as text directly into the file. Remember that when `vi` is in the input mode, you can return `vi` to the command mode by pressing the Escape key once.

There are several different commands used to insert text, and all of the commands that are used to insert text automatically place `vi` in the input mode as soon as the command is typed.

To begin this exercise, open `my.file` and make sure the cursor is positioned at the word `text` in the first line of the file.

As you did initially to insert text into `my.file`, you will use the insert command to insert the word `new` just before the word `text`. With the cursor positioned on the first `t` in the word `text`, put `vi` into the input mode command by typing the insert command:

```
i
```

Next, enter the word `new` and press the space bar once:

```
new Space
```

Exit the input mode by pressing the Escape key:

```
Escape
```

The cursor should now be positioned on the space between the words `new` and `text`.

The `i` command starts inserting text at the character just before the cursor. That's why you have to remember to press the Space bar to insert a space between words if the cursor was positioned at the first character in a word when you started to insert text.

Another command that is used to insert text is the append (`a`) command. In contrast to the insert command, the `a` command adds (or appends) the characters you type just after the cursor position. To see how the `a` command works, use the cursor movement keys to move to the letter `u` in the word `You`, and type:

```
a  
, too, Escape
```

The `vi` text editor appended the text you typed to the end of the word `You`. The cursor should now be positioned on the second comma.

The `o` command opens a new line below the line with the cursor and allows you to insert text at the start of that new line. To add a sentence to the end of this file, move the cursor to the last line of the file by pressing the Return key three times:

```
Return
Return
Return
```

The cursor should be positioned at the word `will`. To open a new line below the current line and automatically put `vi` into the input mode, type:

```
o
```

Enter the sample text shown below (including pressing the Return key where indicated), and press the Escape key to return to command mode when you are finished.

```
New text can be easily entered Return
while in input mode. Escape
```

Your screen should now look like this:

```
You, too, can use this new text file
to experiment with vi.
The examples shown here
will teach you the basics of vi.
New text can be easily entered
while in input mode.
~
~
~
~
~
~
```

The `O` command opens a new line above the current line and starts inserting text at the start of the new line. This command is most useful for adding new text to the top of an existing file, but can be used anywhere in a file. To practice using this command to open a line and insert text, move the cursor to the first line in the file (using the cursor movement command `H` perhaps) and type:

```
O
Opening a new line is easy. Escape
```

The `vi` text editor is back in the command mode once the Escape key is pressed.

There are two other commands that put `vi` in the input mode: the `I` and `A` commands. The `I` command inserts text at the beginning of the current line. The `A` command appends text after the last character at the end of the current line.

Practice inserting text to the beginning of a line, by typing:

```
I
Inserting text is easy. Space  Escape
```

Practice appending text to the end of a line by typing:

```
A
Really! Escape
```

Your screen should now look like this:

```
Inserting a line is easy. Opening a new line is easy. Really!
You, too, can use this new text file
to experiment with vi.
The examples shown here
will teach you the basics of vi.
New text can be easily entered
while in input mode.
~
~
~
~
~
~
```

Table A-3 summarizes the commands used to insert and append text to a file. These commands are executed from the command mode and automatically put `vi` into the input mode.

Table A-3: Text Insertion Command Summary

Command	Result
<code>i</code>	Inserts text immediately before the current cursor position.
<code>a</code>	Appends text immediately after the current cursor position.
<code>I</code>	Inserts text at the beginning of the current line.
<code>A</code>	Appends text to the end of the current line.
<code>o</code>	Opens a new line directly below the current line.
<code>O</code>	Opens a new line directly above the current line.

A.1.6 Editing Text

Up to this point you have only learned how to add new text to the file, but what if you need to change some text? The `vi` text editor provides commands for deleting and changing text. For example, to remove the word `easily`, from the sixth line in `my.file`, move the cursor to the first character of the word and enter:

```
d w
```

This command is a combination of the delete command `d`, and the motion command `w`. In fact, many `vi` commands can be combined with motion commands to specify the duration of the action. The general form of a `vi` command follows:

```
[number] [command] motion
```

The *command* entry represents an action command, *motion* represents a motion command, and *number* optionally represents the number of times to perform the command. You also can use this general form to move the cursor in larger steps.

To illustrate this concept, move the cursor to the beginning of `my.file` by typing `H`. Now, to move the cursor forward four words, enter:

```
4 w
```

The cursor has moved four entire words and is positioned at the first letter of the fifth word, `easy`.

A.1.6.1 Deleting Words

Using the general form of commands, you can delete the last five words of this text file. Move the cursor to the beginning of the last line by pressing the Return key several times and enter:

```
5 d w  
: w
```

It takes five words to delete the whole line rather than four because the period at the end of the line counts as a word. All punctuation counts as one word when you're using the delete word command. As a reminder that you should save a file often, this example also had you `w r i t e` the file (save it) using the `:w` command.

Suppose you only want to delete a portion of a word? The `x` command deletes one character at a time. To see how this command works, move the cursor to the letter `s` in the word `examples`. Press the `x` key once to delete the letter `s`.

A.1.6.2 Deleting Lines

The `dd` command is a shortcut for deleting whole lines at a time. The `dd` command can be used with a number to delete multiple lines as well. For example, position the cursor at the sixth line in the file (at the line beginning with the word `New`) and type:

```
2dd
```

The sixth and seventh lines (even though the seventh line is empty) of the file are deleted simultaneously. The `dd` command can be used without specifying a number to delete one line at a time.

The `D` command clears the current line of text from the current cursor position to the end of the line but does not delete the line itself. If the cursor is positioned at the beginning of the line, the entire line is cleared. This command speeds up your work because you don't have to know how many words are in the line to be able to delete them (as you would, for example, if you were using the `dw` command). This command is useful if you want to rewrite an entire line. With the cursor positioned at the beginning of the line, the `D` command followed by one of the text insertion commands (`i`, `I`, `a`, or `A`) allows you to clear the current line of text and reenter new text with a minimum of keystrokes.

A.1.6.3 Changing Text

The command for changing text, `c`, can be used to combine the actions of deleting and returning to input mode. It follows the same general form as the `d` command. To change the text `new text` to `almost new demo`, move the cursor to the first character in the word "new", and enter the command:

```
2cw
```

The text will not immediately disappear. Instead, a dollar sign (\$) is placed at the end of the change range (the last `t` in `text`), and `vi` is placed in input mode automatically. The text you enter will overwrite the existing text up to the dollar sign and then extend the text range as needed. Enter the new text by typing:

```
almost new demo Escape
```

A.1.6.4 Text Editing Command Summary

As shown in the previous sections, the text editing commands can be used together with the motion commands to give you more editing power. The text editing commands can be combined with a number to change or delete large blocks of words or lines simultaneously. Table A-4 summarizes the commands used to edit text.

Table A-4: Text Editing Command Summary

Command	Result
<code>cw</code>	Changes the current word to the new text you type. You may change the word with as much next text as necessary. The Escape key signals the end of the change.
<code>ncw</code>	Changes <i>n</i> number of words to the new text you type. The new text is not limited to just <i>n</i> words. You may change <i>n</i> words with as much new text as necessary. The Escape key signals the end of the change.
<code>D</code>	Clears the text from the current cursor position to the end of the line. Does not delete the space used by the line thereby allowing you to add more text.
<code>dd</code>	Deletes the current line.
<code>ndd</code>	Deletes <i>n</i> number of lines beginning with the current line.
<code>dw</code>	Deletes the current word.
<code>ndw</code>	Deletes <i>n</i> number of words beginning with the current word.
<code>x</code>	Deletes the current character.

A.1.7 Undoing a Command

If you make a change and then realize it was in error, you may still be able to correct it if you haven't executed another command. The `u` command undoes the last command entered. Undo the last command, `2cw`, by typing:

`u`

The text string `almost new demo` will be changed back to `new text` if you didn't execute any other commands since you executed the `2cw` command.

The uppercase `U` command undoes all changes to the current line and restores the line to its original state. The `U` command works only if you have not moved the cursor to another line.

A.1.8 Finishing Your Edit Session

After you finish the exercises in this appendix, you should save the file and quit `vi`. To save your changes and quit `vi`, enter:

```
:wq Return
```

If you want to quit `vi` without saving your changes, you can do so by entering:

```
:q! Return
```

You have now learned enough about `vi` to edit any file. The following sections show you some advanced techniques that can improve your productivity and allow you to customize your environment.

A.2 Using Advanced Techniques

This section shows you how to search for text strings, move text, and copy and paste text. As you work with larger documents, all these tasks increase your ability to work efficiently.

A.2.1 Searching for Strings

In a large document, searching for a particular text string can be very time consuming. The slash (`/`) command prompts for a text string to search for in the file. When you enter the slash (`/`) and press the Return key, `vi` searches the file for the first occurrence of the text string you entered.

If you do not have it open, reopen the `my.file` file. Move to the top of the document using one of the cursor movement keys you learned earlier in this appendix. To search for the text string `th`, enter the following:

```
/th Return
```

As soon as you enter the slash (`/`) command, the slash (`/`) is displayed at the bottom of the screen (similar to the way in which the colon (`:`) works). When you entered the text string `th`, it was echoed (displayed) at the bottom of the screen. You can use the Backspace key to fix mistakes when you enter the search string.

After you press the Return key, the cursor moves to the first occurrence of the string (the `th` in the word `this`).

The `n` (next) command continues the search for the next occurrence of the last string you searched for. Try it now by entering:

```
n
```

The cursor should move to the next occurrence of the string, which is the `th` in the word `with`.

Similarly, the `N` command searches for the next occurrence of the search string, but it searches in the opposite direction of the `n` command. The `N` finds the previous occurrence of the string.

The question mark (`?`) command is also used to initiate a search for text strings, but the question mark (`?`) initiates a backward search through the file. When you search backward, the `n` command moves the cursor backward to the previous occurrence of the string, and the `N` command moves the cursor forward (exactly the opposite of the way in which they work with a slash (`/`) search).

A.2.2 Deleting and Moving Text

To move a block of text, you must first select the text to move. You already know how to do this. The delete (`d`) command not only deletes a line of text but also copies it to a paste buffer. Once in the paste buffer, the text can be moved (or pasted) by repositioning the cursor and then using the lowercase `p` command to paste the text on the line after the current cursor position.

Move the cursor to the first line in the file and type:

```
dd
```

The line is deleted and copied into the paste buffer, and the cursor is located on the next line in the file. To paste the line in the buffer back into the file, after the line on which the cursor is positioned, enter:

```
p
```

The uppercase letter `P` (Paste) command is used to paste text on the line above the cursor rather than below it.

If you delete a letter or block of words, the deleted text will be pasted into the new position within the current line. For example, to move the word `can` to just before the word `with`, use the following command sequence (remember to use an uppercase `P`):

```
/can Return  
dw  
/with Return  
P
```

A.2.3 Yanking and Moving Text

You copy text in the same manner as you move it, except that instead of using the delete text command `d`, you use the yank text command, `y`. The `y` command copies (or yanks) the specified text into the paste buffer without deleting it from the text. It follows the same syntax as the `d` command. You can also use the `yy` command to yank an entire text line into the paste buffer, in the same way as `dd`.

For example, to copy the first two lines of the file to a position immediately underneath them, enter the following command sequence from the first line of the file:

```
2yy
j
p
```

You must move the cursor down one line using `j` or the two lines will be pasted after the first line rather than after the second.

A.2.4 Other vi Features

You may want to try some of the other features of `vi`. The `vi(1)` reference page lists all of the available commands. You may want to pay particular attention to the following:

<code>J</code>	Joins the next line of text to the current line of text.
<code>.</code>	Repeats the last command.
<code>s</code>	Substitutes the current character with the following entered text.
<code>x</code>	Deletes the current character.
<code>~</code>	Changes the alphabetic case of the current character.
<code>!</code>	Executes an operating system command on the current line of text and replaces the text with the output.
<code>Ctrl/L</code>	Refreshes the screen when problems with the screen display occur. Any time your screen is displaying confusing output, press <code>Ctrl/L</code> .

A.3 Using the Underlying ex Commands

The `vi` text editor is based upon the `ex` line editor. The underlying `ex` line editor can bring the power of global changes to your entire text file or any large piece of it. You can access `ex` commands from within `vi` by using the colon (`:`) command. You were introduced to `ex` commands earlier in this appendix with the `:wq` and `:q!` commands for writing and quitting an editing session.

The colon (`:`) command causes `ex` to prompt for a command line at the bottom of the editor screen with a colon (`:`). Each `ex` command is ended by pressing the Return key. You can also enter `ex` more permanently with the `vi` command `Q`. This command turns processing over to `ex` until you explicitly return to `vi`. This often happens accidentally. If it should happen to you, you can return to `vi` by typing `vi` at the colon (`:`) prompt followed by the Return key as follows:

```
:vi Return
```

An `ex` command acts on a block of lines in your text file according to the following general syntax:

```
:[address[,address]]command
```

The *command*, along with any of its arguments, acts on the lines between and including the first and second *address*. If one address is specified, the command acts only on the specified line. If no address is specified, the command acts only on the current line. Addresses can be specified in a number of ways. Some of the more common address specifications are the following:

<i>line number</i>	Address by absolute line number.
<i>/pattern/</i>	Next line that contains the pattern.
<code>.</code>	Line that the cursor is on.
<code>\$</code>	Last line of the file.
<i>address±lines</i>	Relative offset from the addressed line.
<code>%</code>	All the lines in the file, and is used once in place of both addresses.

The following sections show some of the most generally useful `ex` commands, and some of the customization features offered by `ex`. You should read the `ex(1)` reference page for a more detailed list of commands.

A.3.1 Making Substitutions

The most common substitution task, possibly the most common `ex` task, is a global substitution of one word or phrase for another. You can do this with the `s` command. If you have closed the `my.file` file, reopen it. To change every occurrence of "is" to "was", use the following command:

```
:%s/is/was/g Return
```

This substitution command is applied to all lines in the file by the `%` address. The slash (`/`) is used as a separator. The `g` argument at the end of the command causes the substitution to occur globally, that is, on each instance of the pattern within each line. Without the `g` argument, substitution occurs only once on each line.

You should be careful when making substitutions to ensure that you get what you want. In the previous command line, the word `this` has changed to `thwas` because every occurrence of `is` was changed to `was`.

You can add a `c` argument along with the `g` argument to prompt for confirmation before each substitution. The format of the confirmation is a bit complex; however, it is well worth using when you want to be scrupulous about making global changes.

As an example of confirming a substitution, change the word `thwas` back to `this` by issuing the following command:

```
:%s/thwas/this/gc Return
```

The following prompt appears at the bottom of the screen:

```
You, too, use thwas new text file
      ^^^^^
```

As shown in the next example, type `y` and press the Return key. You are then prompted for the second substitution:

```
You, too, use thwas new text file
      ^^^^^y Return
You, too, use thwas new text file
      ^^^^^
```

Type `y` and press the Return key, and in response to the `Hit return to continue` prompt, press the Return key once again as follows:

```
You, too, use thwas new text file
      ^^^^^y
You, too, use thwas new text file
      ^^^^^y Return
[Hit return to continue] Return
```

You will find that the two occurrences of the word `thwas` have been changed back to `this`. In addition, `vi` is back in the command mode with the cursor positioned at the first character of the line with the last substitution.

Now try another substitution on your example file. Add three lines of new text to the file by using the `$` (go to beginning of last line), `o` (create new line), `yy` (yank), and `p` (paste) commands as follows:

```
:$ Return
o
Some new text with a misspelling. Escape
yy
p
p
p
```

You now should have four lines of new text, all containing the incorrectly spelled word `misspelling`.

To fix the spelling error, enter one of the following commands:

```
:1,$s/misspelling/misspelling/ Return
```

or

```
:5,8s/misspelling/misspelling/ Return
```

In the first example, the address `1,$` indicates that the substitution should begin on line one (1) and end at the last line of the file (`$`). In the second example, `5,8` indicates that the substitution should be on line five and end on line eight. You do not need to use the `g` operator in either case because the change is only necessary once on each line.

A.3.2 Writing a Whole File or Parts of a File

The `:wq` command is a special `ex` command that writes the whole file. It combines the features of the write command `w` and the quit command `q`. The only argument that the quit command can take is the exclamation point (!). It forces the session to quit even if changes made to the file would be lost by quitting.

The `w` command can also take addresses and a filename argument, which allows you to save part of your text to another file. For example, to save the first three lines of your text to the new file `my.new.file`, use the following command:

```
:1,3w my.new.file Return
```

```
"my.new.file" [New file] 3 lines, 130 characters
```

A.3.3 Deleting a Block of Text

The delete command in `ex` is `d`, just as in `vi`. To delete from the current line to the end of the file, use the following command:

```
:.,$d Return
```

A.3.4 Customizing Your Environment

The `ex` editor provides two mechanisms for customizing your `vi` environment. You can use the `:set` command to set environment variables, and the `:map` command to map a key sequence to a `vi` command key.

Environment variables are set either by assigning them as *option* or *no option* for Boolean variables, or by assigning them as *option=value*. The full set of environment variables is described in the `ex(1)` reference page. Table A-5 lists some common variables.

Table A-5: Selected vi Environment Variables

Variable	Description
<i>errorbells</i>	Specifies that when an error is made, a bell sounds. This is the default setting.
<i>ignorecase</i>	Specifies that when performing searches, the case of characters should be ignored. The default variable setting is <i>noignorecase</i> .
<i>number</i>	Specifies that line numbers are to be displayed at the left margin. The default variable setting is <i>nonumber</i> .
<i>showmatch</i>	Specifies that when you enter a matching parenthesis or brace, the cursor moves to the matching character and then returns. The default variable setting is <i>noshowmatch</i> .
<i>tabstop</i>	Specifies the amount of space between tab stops. The default setting is 8.
<i>wrapscan</i>	Specifies that searches should wrap around the beginning or end of the file. This is the default variable setting.

Table A-5: (continued)

Variable	Description
<i>wrapmargin</i>	<p>Creates an automatic right margin located a specified number of characters from the right side of your screen. Whenever your cursor reaches the specified right margin, an automatic new line is generated, and the word you are typing is brought to the next line.</p> <p>We recommend that you set the <i>wrapmargin</i> variable to a value with which you are comfortable. Otherwise, <i>vi</i> will use the default setting of 0. Using the default setting means that your cursor jumps to the next line when it reaches the end of your screen; however, parts of the word you are keying may be on separate lines.</p>

To display the line numbers of your example file enter the following command:

```
:set number Return
```

To remove the line numbers, enter the following command:

```
:set nonumber Return
```

The `:map` command sets a single *vi* command key to a *vi* command sequence. The syntax for the `:map` command follows:

```
:map key sequence Return
```

This command sequence replaces any existing command for that key. The command sequence should be identical to the keystrokes you want to map, except that special keys such as the Return key, the Escape key, and keys modified with the Ctrl key must be quoted first with Ctrl/V. Because the `q` and `v` keys do not have commands associated with them, they are good keys to map.

For example, to map a key sequence that inserts a line into your text that says "This space held for new text", you could use the following command:

```
:map q oThis space held for new text Ctrl/V Escape Return
```

Note the use of Ctrl/V to quote the Escape character.

A.3.5 Saving Your Customizations

You can make your environment customizations permanent by placing the appropriate `ex` commands in a file named `.exrc` in your home directory. Commands in this file will take effect every time you enter `vi` or `ex`. In this file, you do not need to use the `vi` command `:`, because these commands are read directly by the underlying `ex` editor.

For example, to customize your environment to always display line numbers for your files, to use the map sequence shown in the previous section, and to set an automatic right margin of five spaces, you would first open the `.exrc` file with `vi` in your home directory, and add the following lines of text:

```
set number
map q oThis space held for new text Ctrl/V Escape
set wrapmargin=5
```

After you write this file, verify that it works by opening your example file.

Creating and Editing Files with ed **B**

This appendix explains how to create, edit (modify), display, and save text files with `ed`, a line editing program. If your system has another editing program, you may want to learn how to do these tasks with that program.

A good way to learn how `ed` works is to try the examples in this appendix on your system. Since the examples build upon each other, it is important for you to work through them in sequence. Also, to make what you see on the screen consistent with what you see in this guide, it is important to do the examples just as they are given.

In the examples, everything you should enter is printed in **boldface**. When you are told in the text to enter something, you should enter all of the information for that line and then press Return.

Because `ed` is a line editor, you can work with the contents of a file only one line at a time. Regardless of what text is on the screen, you can edit only the current line. If you have experience with a screen editing program, you should pay careful attention to the differences between that program and `ed`. For example, with the `ed` program, you cannot use the Cursor Up and Cursor Down keys to change your current line.

B.1 Understanding Text Files and the Edit Buffer

A file is a collection of data stored together in the computer under an assigned name. You can think of a file as the computer equivalent of an ordinary file folder – it may contain the text of a letter, a report, or some other document, or the source code for a computer program.

The edit buffer is a temporary storage area that holds a file while you work with it – the computer equivalent of the top of your desk. When you work with a text file, you place it in the edit buffer, make your changes to the file (edit it), and then transfer (copy) the contents of the buffer to a permanent storage area.

The rest of this appendix explains how to create, display, save, and edit (modify) text files with the `ed` editor.

B.2 Creating and Saving Text Files

To create and save a text file, perform the following steps. The following sections describe these steps in detail.

1. At the shell prompt, enter the following command:

```
$ ed filename
```

The *filename* argument is the name of the file you want to create or edit.

2. When you receive the ? *filename* message, enter the following append command:

```
a
```

3. Enter your text.

4. To stop adding text, enter a dot (.) at the start of a new line.

5. To copy the contents of the edit buffer into the *filename* file, enter the following command:

```
w
```

6. To end the ed program, enter the following command:

```
q
```

B.2.1 Starting the ed Program

To start the ed program, enter a command of the form `ed filename` after the shell prompt (\$).

In the following example, the `ed afile` command starts the ed program and indicates that you want to work with a file named `afile`:

```
$ ed afile
?afile
-
```

The ed program responds with the message `?afile`, which means that the file does not exist. You can now use the `a` (append) subcommand (described in the next section) to create `afile` and put text into it.

B.2.2 Entering Text – The a (Append) Subcommand

To put text into your file, enter `a`. The `a` subcommand tells ed to add, or append, the text you enter to the edit buffer. If your file had already contained text, the `a` subcommand would add the new text to the end of the file.

Type your text, pressing Return at the end of each line. When you have entered all of your text, enter a dot (.) at the start of a new line.

Note

If you do not press Return at the end of each line, the `ed` program automatically moves your cursor to the next line after you fill a line with characters. However, `ed` treats everything you enter before you press Return as one line, regardless of how many lines it takes up on the screen; that is, the line wraps around to the beginning of the next line (based upon your workstation display settings).

The following example shows how to enter text into the `afile` file:

```
a
The only way to stop
appending is to enter a
line that contains only
a dot.
```

```
·
—
```

If you stop adding text to the buffer and then decide to add some more, enter another a subcommand. Type the text and then enter a dot at the start of a new line to stop adding text to the buffer.

If you make errors as you enter your text, you can correct them before you press Return. Use the Backspace key to erase the incorrect character(s). Then enter the correct characters in their place.

B.2.3 Displaying Text – The `p` (print) Subcommand

Use the `p` (print) subcommand to display the contents of the edit buffer.

To display a single line, use the `np` subcommand, where *n* is the number of the line. For example:

```
2p
appending is to enter a
```

```
—
```

To display a series of lines, use the `n,mp` subcommand, where *n* is the starting line number and *m* is the ending line number. For example:

```
1,3p
The only way to stop
appending is to enter a
line that contains only
```

```
—
```

To display everything from a specific line to the end of the buffer, use the `n,$p` subcommand, where *n* is the starting line number and `$` stands for the

last line of the buffer. In the following example, `1, $p` displays everything in the buffer:

```
1,$p
The only way to stop
appending is to enter a
line that contains only
a dot.
-
```

Note

Many examples in the rest of this appendix use `1, $p` to display the buffer's contents. In these examples, the `1, $p` subcommand is optional and convenient – it lets you verify that the subcommands in examples work as they should. Another convenient `ed` convention is `,p`, which is equivalent to `1, $p` – that is, it displays the contents of the buffer.

B.2.4 Saving Text – The `w` (write) Subcommand

The `w` (write) subcommand writes, or copies, the contents of the buffer into a file. You can save all or part of a file under its original name or under a different name. In either case, `ed` replaces the original contents of the file you specify with the data copied from the buffer.

B.2.4.1 Saving Text under the Same Filename

To save the contents of the buffer under the original name for the file, enter the `w` subcommand. For example:

```
w
78
-
```

The `ed` program copies the contents of the buffer into the file named `afile` and displays the number of characters copied into the file (78). This number includes blanks and characters such as Return (sometimes called newline), which are not visible on the screen.

The `w` subcommand does not affect the contents of the edit buffer. You can save a copy of the file and then continue to work with the contents of the buffer.

The stored file is not changed until the next time you use the `w` subcommand to copy the contents of the buffer into it. As a safeguard, it is a good practice to save a file periodically while you work on it. Then, if you make changes (or mistakes) that you do not want to save, you can start over with the most recently saved version of the file.

Note

The `u` (undo) subcommand restores the buffer to the state it was in before it was last modified by an `ed` subcommand. The subcommands that `u` can reverse are `a`, `c`, `d`, `g`, `G`, `i`, `j`, `m`, `r`, `s`, `t`, `v`, and `V`.

B.2.4.2 Saving Text under a Different Filename

Often, you may need more than one copy of the same file. For example, you could have the original text of a letter in two files – one to keep as it is, and the other to be revised.

If you have followed the previous examples, you have a file named `afile` that contains the original text of your document. To create another copy of the file (while its contents are still in the buffer), use a subcommand of the form `w filename`, as the following example shows:

```
w bfile
78
-
```

At this point, `afile` and `bfile` have the same contents, since each is a copy of the same buffer contents. However, because `afile` and `bfile` are separate files, you can change the contents of one without affecting the contents of the other.

B.2.4.3 Saving Part of a File

To save part of a file, use a subcommand of the form `n,mw filename`, where:

<code>n</code>	Specifies the beginning line number of the part of the file you want to save.
<code>m</code>	Specifies the ending line number of the part of the file you want to save (or the number of a single line, if that is all you want to save).
<code>filename</code>	Specifies the name of a different file (optional).

In the following example, the `w` subcommand copies lines 1 and 2 from the buffer into a new file named `cfile`:

```
1,2w cfile
44
-
```

Then `ed` displays the number of characters written into `cfile` (44).

B.2.5 Leaving the ed Program – The q (quit) Subcommand

Caution

The contents of the buffer are lost when you leave the `ed` program. To save a copy of the data in the buffer, use the `w` subcommand to copy the buffer into a file before you leave the `ed` program.

To leave the `ed` program, enter the `q` (quit) subcommand. For example:

```
q
$ _
```

The `q` subcommand returns you to the shell prompt (`$`).

If you have changed the buffer but have not saved a copy of its contents, the `q` subcommand responds with `?`, an error message. At that point, you can either save a copy of the buffer with the `w` subcommand, or enter `q` again to leave the `ed` program without saving a copy of the buffer.

B.3 Loading Files into the Edit Buffer

Before you can edit a file, you must load it into the edit buffer. You can load a file either at the time you start the `ed` program or while the program is running.

To load a file into the edit buffer when you start the `ed` program, enter the following command:

```
ed filename
```

This command starts `ed` and loads the `filename` file into the edit buffer.

To load a file into the edit buffer while `ed` is running, you can enter one of the following commands:

- `e filename`

This loads the `filename` file into the buffer, erasing any previous contents of the buffer.

- `nr filename`

This reads the `filename` file into the buffer after line `n`. If you do not specify `n`, `ed` adds the file to the end of the buffer.

B.3.1 Using the ed (edit) Command

To load a file into the edit buffer when you start the `ed` program, enter the name of the file after the `ed` command. The `ed` command in the following example invokes the `ed` program and loads the file `afile` into the edit buffer:

```
$ ed afile
78
```

—

The `ed` program displays the number of characters that it read into the edit buffer (78).

If `ed` cannot find the file, it displays `?filename`. To create that file, use the `a` (append) subcommand (described in Section B.2.2) and the `w` (write) subcommand (described in Section B.2.4).

B.3.2 Using the e (Edit) Subcommand

Once you start the `ed` program, you can use the `e` (edit) subcommand to load a file into the buffer. The `e` subcommand replaces the contents of the buffer with the new file. (Compare the `e` subcommand with the `r` subcommand, described next in Section B.3.3, which adds the new file to the buffer.)

Caution

When you load a new file into the buffer, the new file replaces the buffer's previous contents. Save a copy of the buffer with the `w` subcommand before you read a new file into the buffer.

In the following example, the `e cfile` subcommand reads the `cfile` file into the edit buffer, replacing `afile`. The `e afile` subcommand then loads `afile` back into the buffer, deleting `cfile`. The `ed` program returns the number of characters read into the buffer after each `e` subcommand (44 and 78):

```
e cfile
44
e afile
78
```

—

If `ed` cannot find the file, it returns `? filename`. To create that file, use the `a` (append) subcommand, described in Section B.2.2, and the `w` (write) subcommand, described in Section B.2.4.

You can edit any number of files, one at a time, without leaving the `ed` program. Use the `e` subcommand to load a file into the buffer, make your changes to the file, and use the `w` subcommand to save a copy of the revised file. (See Section B.2.4 for information about the `w` subcommand.) Then use the `e` subcommand again to load another file into the buffer.

B.3.3 Using the `r` (read) Subcommand

Once you have started the `ed` program, you can use the `r` (read) subcommand to read a file into the buffer. The `r` subcommand adds the contents of the file to the contents of the buffer. The `r` subcommand does not delete the buffer. (Compare the `r` subcommand with the `e` subcommand, described in Section B.3.2, which deletes the buffer before it reads in another file.)

With the `r` subcommand, you can read a file into the buffer at a particular place. For example, the `4r cfile` subcommand reads the file `cfile` into the buffer following line 4. The `ed` program then renumbers all of the lines in the buffer. If you do not use a line number, the `r` subcommand adds the new file to the end of the buffer's contents.

The following example shows how to use the `r` subcommand with a line number:

```
1,$p
The only way to stop
appending is to enter a
line that contains only
a dot.
3r cfile
44
1,$p
The only way to stop
appending is to enter a
line that contains only
The only way to stop
appending is to enter a
a dot.
-
```

The `1,$p` subcommand displays the four lines of `afile`. Next, the `3r cfile` subcommand loads the contents of `cfile` into the buffer, following line 3, and shows that it read 44 characters into the buffer. The next `1,$p` subcommand displays the buffer's contents again, letting you verify that the `r` subcommand read `cfile` into the buffer after line 3.

If you are working the examples on your system, complete the following steps before you go to the next section:

1. Save the contents of the buffer in the `cfile` file:

```
w cfile
```

2. Load `afile` into the buffer:

```
e afile
```

B.4 Displaying and Changing the Current Line

The `ed` program is a **line editor**. This means that `ed` lets you work with the contents of the buffer one line at a time. The line you can work with at any given time is called the "current line", and it is represented by the dot (`.`). To work with different parts of a file, you must change the current line.

To display the current line, enter the following subcommand:

```
P
```

To display the line number of the current line, enter the following subcommand:

```
.=
```

Note

You cannot use the Cursor Up and Cursor Down keys to change the current line. To change the current line, use the `ed` subcommands described in the following sections.

To change your position in the buffer, do one of the following. These steps are described in detail in the following sections.

1. To set your current line to line number n , enter the following subcommand:

```
n
```

2. To move the current line forward through the buffer one line at a time press Return.
3. To move the current line backward through the buffer one line at a time, enter a dash (`-`) character.
4. To move the current line n lines forward through the buffer, enter the following subcommand:

```
."+n
```

5. To move the current line *n* lines backward through the buffer, enter the following subcommand:

```
.-n
```

B.4.1 Finding Your Position in the Buffer

When you first load a file into the buffer, the last line of the file is the current line. As you work with the file, you usually change the current line many times. You can display the current line or its line number at any time.

To display the current line, enter `p`:

```
p
a dot .
-
```

The `p` subcommand displays the current line (`a dot.`). Because the current line has not been changed since you read `afile` into the buffer, the current line is the last line of the buffer.

Enter `. =` to display the line number of the current line:

```
. =
4
-
```

Since `afile` has four lines, and the current line is the last line in the buffer, the `. =` subcommand displays 4.

You also can use the dollar sign (the symbol that stands for the last line in the buffer) with the `=` subcommand to determine the number of the last line in the buffer:

```
$ =
4
-
```

The `$ =` subcommand is an easy way to find out how many lines are in the buffer. The `ed` `$` symbol has no relationship to the shell prompt (`$`).

B.4.2 Changing Your Position in the Buffer

You can change your position in the buffer (change your current line) in one of two ways:

- Specify a line number (an absolute position)
- Move forward or backward relative to your current line

To move the current line to a specific line, enter the line number; `ed` displays the new current line. In the following example, the first line of a file becomes the current line:

```
1
The only way to stop
```

-

Pressing Return advances one line through the buffer and displays the new current line, as the following example shows:

```
appending is to enter a
line that contains only
a dot.
```

?

-

When you try to move beyond the last line of the buffer, `ed` returns `?`, an error message. You cannot move beyond the end of the buffer.

To set the current line to the last line of the buffer, enter `$`.

To move the current line backward through the buffer one line at a time, enter dashes (`-`) one after the other, as the following example shows:

```
-
line that contains only
-
appending is to enter a
-
The only way to stop
-
?_
```

When you try to move beyond the first line in the buffer, you receive the `?` message. You cannot move beyond the top of the buffer.

To move the current line forward through the buffer more than one line at a time, enter `.n` (where `n` is the number of lines you want to move):

```
.2
line that contains only
```

-

Note that `.2` is an abbreviation for `+.2`.

To move the current line backward through the buffer more than one line at a time, enter the following subcommand: `. - n` (where *n* is the number of lines you want to move):

```
.-2
The only way to stop
-
```

B.5 Locating Text

If you do not know the number of the line that contains a particular word or another string of characters, you can locate the line with a context search.

To search in context, do one of the following:

- To search forward, enter the following subcommand:
`/string to find/`
- To search backward, enter the following subcommand:
`?string to find?`

The following sections describe these methods of searching text in detail.

B.5.1 Searching Forward Through the Buffer

To search forward through the buffer, enter the string enclosed in slashes (`//`):

```
/only/
line that contains only
-
```

The context search (`/only/`) begins on the first line after the current line, then locates and displays the next line that contains the string "only". That line becomes the current line.

If `ed` does not find the string between the first line of the search and the last line of the buffer, then it continues the search at line 1 and searches to the current line. If `ed` searches the entire buffer without finding the string, it displays the `? error` message:

```
/random/
?
-
```

Once you have searched for a string, you can search for the same string again by entering `//`. The following example shows one search for the string

only, and then a second search for the same string:

```
/only/  
The only way to stop  
//  
line that contains only  
-
```

B.5.2 Searching Backward Through the Buffer

Searching backward through the buffer is much like searching forward, except that you enclose the string in question marks (?):

```
?appending?  
appending is to enter a  
-
```

The context search begins on the first line before the current line, and locates the first line that contains the string `appending`. That line becomes the current line. If `ed` searches the entire buffer without finding the string, it stops the search at the current line and displays the message `?`.

Once you have searched backward for a string, you can search backward for the same string again by entering `??`. This is because `ed` remembers search strings.

B.5.3 Changing the Direction of a Search

You can change the direction of a search for a particular string by using the slash (/) and question mark (?) search characters alternately:

```
/only/  
line that contains only  
??  
The only way to stop  
-
```

If you go too far while searching for a character string, it is convenient to be able to change the direction of your search.

B.6 Making Substitutions – The `s` (substitute) Subcommand

Use the `s` (substitute) subcommand to replace a character string (a group of one or more characters) with another. The `s` subcommand works with one or more lines at a time, and is especially useful for correcting typing or spelling errors.

To make substitutions, do one of the following:

- To substitute *newstring* for *oldstring* at the first occurrence of *oldstring* in the current line, enter the following subcommand:
s/oldstring/newstring/
- To substitute *newstring* for *oldstring* at the first occurrence of *oldstring* on line number *n*, enter the following subcommand:
ns/oldstring/newstring/
- To substitute *newstring* for *oldstring* at the first occurrence of *oldstring* in each of the lines *n* through *m*, enter the following subcommand:
n,ms/oldstring/newstring/

The following sections describe these methods of substitution in detail.

B.6.1 Substituting on the Current Line

To make a substitution on the current line, first make sure that the line you want to change is the current line. In the following example, the `/appending/` (search) subcommand locates the line to be changed. Then the `s/appending/adding text/p` (substitute) subcommand substitutes the string "adding text" for the string "appending" on the current line. The `p` (print) subcommand displays the changed line.

```
/appending/
appending is to enter a
s/appending/adding text/p
adding text is to enter a
—
```

Note

For convenience, you can add the `p` (print) subcommand to the `s` subcommand (for example, `s/appending/adding text/p`). This saves you from having to enter a separate `p` subcommand to see the result of the substitution.

The `s` subcommand changes only the first occurrence of the string on a given line. To learn how to change all occurrences of a string on the line, see Section B.6.4.

B.6.2 Substituting on a Specific Line

To make a substitution on a specific line, use a subcommand of the following form:

```
ns/oldstring/newstring/
```

Here *n* is the number of the line on which the substitution is to be made. In the following example, the *s* subcommand moves to line number 1 and replaces the string "stop" with the string "quit" and displays the new line:

```
1s/stop/quit/p  
The only way to quit
```

—

The *s* subcommand changes only the first occurrence of the string on a given line. To learn how to change all occurrences of a string on the line, see Section B.6.4.

B.6.3 Substituting on Multiple Lines

To make a substitution on multiple lines, use a subcommand of the following form:

```
n,ms/oldstring/newstring/
```

Here *n* is the first line of the group and *m* is the last. In the following example, the *s* subcommand replaces the first occurrence of the string "to" with the string "TO" on every line in the buffer:

```
1,$s/to/TO/  
1,$p  
The only way TO quit  
adding text is TO enter a  
line that contains only  
a dot.
```

—

The *1, \$p* subcommand displays the contents of the buffer, which lets you verify that the substitutions were made.

B.6.4 Changing Every Occurrence of a String

Ordinarily, the *s* (substitute) subcommand changes only the first occurrence of a string on a given line. However, the *g* (global) operator lets you change every occurrence of a string on a line or in a group of lines.

To make a global substitution on a single line, use a subcommand of the following form:

```
ns/oldstring/newstring/
```

In the following example, `3s/on/ON/gp` changes each occurrence of the string "on" to "ON" in line 3 and displays the new line:

```
3s/on/ON/gp
line that cONTains ONly
```

—

To make a global substitution on multiple lines, specify the group of lines with a subcommand of the form:

```
n,msl oldstringl newstringlg
```

In the following example, `1,$s/TO/to/g` changes the string "TO" to the string "to" in every line in the buffer:

```
1,$s/TO/to/g
1,$p
The only way to quit
adding text is to enter a
line that cONTains ONly
a dot.
```

—

B.6.5 Removing Characters

You can use the `s` (substitute) subcommand to remove a string of characters (that is, to replace the string with nothing). To remove characters, use a subcommand of the form `s/oldstring//` (with no space between the last two / characters).

In the following example, `ed` removes the string "adding" from line number 2 and then displays the changed line:

```
2s/adding//
text is to enter a
```

—

B.6.6 Substituting at Line Beginnings and Ends

Two special characters let you make substitutions at the beginning or end of a line:

- `^` (circumflex) Makes a substitution at the beginning of the line.
- `$` (dollar sign) Makes a substitution at the end of the line. (In this context, the dollar sign (\$) character does not stand for the last line in the buffer.)

To make a substitution at the beginning of a line, use the *s/^/newstring* subcommand. In the following example, one *s* subcommand adds the string "Remember" to the start of line number 1. Another *s* subcommand adds the string "adding" to the start of line 2:

```
1s/^/Remember,/p
Remember, The only way to quit
2s/^/adding/p
adding text is to enter a
-
```

To make a substitution at the end of a line, use a subcommand of the form *s/\$/newstring*. In the following example, the *s* subcommand adds the string "Then press Enter." to the end of line number 4:

```
4s/$/ Then press Enter./p
a dot. Then press Enter.
-
```

Notice that the substituted string includes two blanks before the word *Then* to separate the two sentences.

B.6.7 Using a Context Search

If you do not know the number of the line you want to change, you can locate it with a context search. See Section B.5 for more information on context searches.

For convenience, you can combine a context search and a substitution into a single subcommand in the following format:

```
/string to find/s/oldstring/newstring/
```

In the following example, *ed* locates the line that contains the string ", The" and replaces that string with ", the":

```
/, The/s/, The/, the/p
Remember, the only way to quit
-
```

Also, you can use the search string as the string to be replaced with a subcommand of the form */string to find/s//newstring/*. In the following example, *ed* locates the line that contains the string "cONTains ONly", replaces that string with "contains only", and prints the changed line:

```
/cONTains ONly/s//contains only/p
line that contains only
-
```

B.7 Deleting Lines – The d (delete) Subcommand

Use the `d` (delete) subcommand to remove one or more lines from the buffer. The general form of the `d` subcommand is the following:

*starting line,ending line***d**

After you delete lines, `ed` sets the current line to the first line following the lines that were deleted. If you delete the last line from the buffer, the last remaining line in the buffer becomes the current line. After a deletion, `ed` renumbers the remaining lines in the buffer.

To delete lines from the buffer, do the following:

- To delete the current line, enter the following subcommand:
`d`
- To delete line number *n* from the buffer, enter the following subcommand:
`nd`
- To delete lines numbered *n* through *m* from the buffer, enter the following subcommand:
`n,m`**d**

The following sections describe these methods of deleting lines in detail.

B.7.1 Deleting the Current Line

If you want to delete the current line, enter `d`. In the following example, the `1,$p` subcommand displays the entire contents of the buffer, and the `$` subcommand makes the last line of the buffer the current line:

```
1,$p
Remember, the only way to quit
adding is to enter a
line that contains only
a dot. Then press Enter.
$
a dot. Then press Enter
d
-
```

The `d` subcommand then deletes the current line (in this case, the last line in the buffer).

B.7.2 Deleting a Specific Line

If you know the number of the line you want to delete, use a subcommand of the form `nd` to make the deletion. In the following example, the `2d` subcommand deletes line 2 from the buffer:

```
2d
1,$p
Remember, the only way to quit
line that contains only
—
```

The `1, $p` subcommand displays the contents of the buffer, showing that the line was deleted.

B.7.3 Deleting Multiple Lines

To delete a group of lines from the buffer, use a subcommand of the form `n, md`, where `n` is the starting line number and `m` is the ending line number of the group to be deleted.

In the following example, the `1, 2d` subcommand deletes lines 1 and 2:

```
1,2d
1,$p
?
—
```

The `1, $p` subcommand displays the `?` message, indicating that the buffer is empty.

If you are following the examples on your system, you should restore the contents of the buffer before you move on to the next section. The following example shows you how to restore the contents of the buffer:

```
e afile
?
e afile
78
—
```

This command sequence reads a copy of the original file `afile` into the buffer.

B.8 Moving Text – The `m` (move) Subcommand

Use the `m` (move) subcommand to move a group of lines from one place to another in the buffer. After a move, the last line moved becomes the current line.

To move text, enter a subcommand of the following form:

```
x,ymz
```

The *x* variable is the first line of the group to be moved. The *y* variable is the last line of the group to be moved. The *z* variable is the line the moved lines are to follow.

In the following example, the `1,2m4` subcommand moves the first two lines of the buffer to the position following line 4:

```
1,2m4
1,$p
line that contains only
a dot.
The only way to stop
appending is to enter a
—
```

The `1,$p` subcommand displays the contents of the buffer, showing that the move is complete.

To move a group of lines to the top of the buffer, use zero (0) as the line number for the moved lines to follow. In the next example, the `3,4m0` subcommand moves lines 3 and 4 to the top of the buffer:

```
3,4m0
1,$p
The only way to stop
appending is to enter a
line that contains only
a dot.
—
```

The `1,$p` subcommand displays the contents of the buffer, showing that the move was made.

To move a group of lines to the end of the buffer, use `$` as the line number for the moved lines to follow:

```
1,2m$
1,$p
line that contains only
a dot.
The only way to stop
appending is to enter a
—
```

B.9 Changing Lines of Text – The `c` (change) Subcommand

Use the `c` (change) subcommand to replace one or more lines with one or more new lines. The `c` subcommand first deletes the line(s) you want to replace and then lets you enter the new lines, just as if you were using the `a` (append) subcommand. When you have entered all of the new text, enter a dot (.) on a line by itself.

The general form of the `c` subcommand is:

*starting line,ending line***c**

To change lines of text, do the following:

1. Enter a subcommand of the following form:

*n,m***c**

The *n* variable specifies the number of the first line of the group to be deleted. The *m* variable specifies the number of the last line of the group (or the only line) to be deleted.

2. Type the new line(s), pressing Return at the end of each line.
3. Enter a dot on a line by itself.

The following sections describe these methods of searching text in detail.

B.9.1 Changing a Single Line of Text

To change a single line of text, use only one line number with the `c` (change) subcommand. You can replace the single line with as many new lines as you like.

In the following example, the `2c` subcommand deletes line 2 from the buffer, and then you can enter new text:

```
2c
appending new material is to
use the proper keys to create a
.
1,$p
The only way to stop
appending new material is to
use the proper keys to create a
line that contains only
a dot.
```

—

The dot on a line by itself stops `ed` from adding text to the buffer. The `1,$p` subcommand displays the entire contents of the buffer, showing that the change was made.

B.9.2 Changing Multiple Lines of Text

To change more than one line of text, give the starting and ending line numbers of the group of lines to be with the `c` subcommand. You can replace the group of lines with one or more new lines.

In the following example, the `2,3c` subcommand deletes lines 2 and 3 from the buffer, and then you can enter new text:

```
2,3c
adding text is to enter a
.
1,$p
The only way to stop
adding text is to enter a
line that contains only
a dot.
-
```

The dot on a line by itself stops `ed` from adding text to the buffer. The `1,$p` subcommand displays the entire contents of the buffer, showing that the change was made.

B.10 Inserting Text – The `i` (insert) Subcommand

Use the `i` (insert) subcommand to insert one or more new lines of text into the buffer. To locate the place in the buffer for the lines to be inserted, you can use either a line number or a context search. The `i` subcommand inserts new lines before the specified line. (Compare the `i` subcommand with the `a` subcommand, explained in Section B.2.2, which inserts new lines after the specified line.) To insert text, do the following:

1. Enter a subcommand of one of the following types:

```
ni
```

The *n* variable specifies the number of the line the new lines will be inserted above.

```
/string/i
```

The *string* variable specifies a group of characters contained in the line the new lines will be inserted above.

2. Enter the new lines.
3. Enter a dot at the start of a new line.

The following sections describe these methods of inserting text in detail.

B.10.1 Using Line Numbers

If you know the number of the line where you want to insert new lines, you can use an insert subcommand of the form `ni` (where *n* is a line number). The new lines you enter go into the buffer before line number *n*. To end the `i` subcommand, enter a dot (`.`) on a line by itself.

In the following example, the `1, $p` subcommand prints the contents of the buffer. Then the `4i` subcommand inserts new lines before line number 4:

```
1, $p
The only way to stop
adding text is to enter a
line that contains only
a dot.
4i
--repeat, only--
.
1, $p
The only way to stop
adding text is to enter a
line that contains only
--repeat, only--
a dot.
-
```

After `4i`, you enter the new line of text and enter a dot on the next line to end the `i` subcommand. A second `1, $p` subcommand displays the contents of the buffer again, showing that the new text was inserted.

B.10.2 Using a Context Search

Another way to specify where the `i` subcommand inserts new lines is to use a context search. With a subcommand of the form `/string/i`, you can locate the line that contains `string` and insert new lines before that line. When you finish inserting new lines, enter a dot on a line by itself.

In the following example, the `/dot/i` subcommand inserts new text before the line that contains the string "dot":

```
/dot/i
and in the first position--
.
1, $p
The only way to stop
adding text is to enter a
line that contains only
--repeat, only--
and in the first position--
a dot.
-
```

The `1, $p` subcommand displays the entire contents of the buffer, showing the new text.

B.11 Copying Lines – The `t` (transfer) Subcommand

With the `t` (transfer) subcommand, you can copy lines from one place in the buffer and insert the copies elsewhere. The `t` subcommand does not affect the original lines.

The general form of the `t` subcommand is:

starting line,ending line`t`*line to follow*

To copy lines, enter a subcommand of the form:

n,m`t`*x*

The *n* variable specifies the first line of the group to be copied. The *m* variable specifies the last line of the group to be copied. The *x* variable specifies the line the copied lines are to follow.

To copy lines to the top of the buffer, use zero (0) as the line number for the copied lines to follow. To copy lines to the bottom of the buffer, use the dollar sign (\$) as the line number for the copied lines to follow.

In the following example, the `1,3t4` subcommand copies lines 1 through 3, and inserts the copies after line 4:

```
1,3t4
```

```
1,$p
```

```
The only way to stop
adding text is to enter a
line that contains only
--repeat, only--
The only way to stop
adding text is to enter a
line that contains only
and in the first position--
a dot.
```

```
-
```

The `1,$p` subcommand displays the entire contents of the buffer, showing that `ed` has made and inserted the copies, and that the original lines are not affected.

B.12 Using System Commands from `ed`

Sometimes you may find it convenient to use a system command without leaving the `ed` program. At these times you can use the exclamation point (!) character to leave the `ed` program temporarily.

To use a system command from `ed`, enter the following:

```
!command
```


In the following example, the `!ls` command temporarily suspends the `ed` program and runs the `ls` (list) system command (a command that lists the files in the current directory):

```
!ls
afile
bfile
cfile
!
-
```

The `ls` command displays the names of the files in the current directory (`afile`, `bfile`, and `cfile`), and then displays another `!` character. The `ls` command is finished, and you can continue to use `ed`.

You can use any system command from within the `ed` program. You can even run another `ed` program, edit a file, and then return to the original `ed` program. From the second `ed` program, you can run a third `ed` program, use a system command, and so forth.

B.13 Ending the `ed` Program

This completes the introduction to the `ed` program. To save your file and end the `ed` program, perform the following steps:

1. Enter the `w` command, as follows:

```
w
```

2. Enter the `q` command, as follows:

```
q
```

For a full discussion of the `w` and `q` subcommands, see Section B.2.4 and Section B.2.5 respectively.

For information about other features of `ed`, see the `ed(1)` reference page.

For information about printing the files you create with `ed`, see Chapter 3.

Using Internationalization Features

C

This appendix describes the internationalization features of the operating system. These features provide users with the ability to process data and to interact with the system in a manner appropriate to their native language, customs, and geographic region (their locale).

After reading this appendix, you will be able to do the following:

- Understand the concept of locale
- Understand what functions are affected by locale
- Determine whether a locale has been set (if necessary)
- Set your locale (if necessary)
- Change your locale or aspects of your locale (if necessary)

If your site is in the United States and you plan to use the American English language and its conventions, there is no need to set a locale because the system default is American English.

If your site is outside the United States, the locale will most likely have already been specified by the system administrator. If the locale has already been set, you may want to only skim this appendix for background information on internationalization. If the locale has not been set, the information in this appendix is essential to you.

C.1 Understanding Locale

Because Digital UNIX is an internationalized operating system, it can present information in a variety of ways. Users tell the operating system how to process and present information in a way appropriate for their language, country, and cultural customs by specifying a locale. See Section C.4 for information about how to specify a locale.

A locale generally consists of three parts: language, territory, and codeset. All three are important for specifying how information is processed and displayed:

- Language specifies the native language (for example, German, French, English).

- Territory specifies the geographic area (for example, Germany, France, Great Britain).
- Codeset specifies the coded character set that is used for the locale (for example, ISO 8859/1, the ISO Latin-1 codeset).

At this point, some background information about codesets may be helpful.

The ASCII codeset has traditionally been used on UNIX systems to express American English. Each letter of the English alphabet (A to Z, a to z) as well as digits, control characters, and symbols are uniquely identified using only 7 of the 8 bits in a standard byte. However, the introduction of new codesets or expansion of old ones has been necessary to include non-English characters. Because so many programs rely on ASCII characters in one way or another, the most commonly used codesets begin with ASCII and build from there.

By using all 8 bits of a standard byte, a single codeset can uniquely identify characters in several alphabetic languages. The most popular codesets are a series called ISO 8859. The first in the series is called ISO 8859/1, the second is ISO 8859/2, and so on through ISO 8859/10. The ISO 8859/1 codeset, often called Latin-1, supports English and other Western European languages.

To identify all ideographic symbols in Asian languages, such as Chinese and Japanese, character encoding requires more than one byte. Numerous codesets using multibyte character encoding, which is not supported by the ISO 8859 series of codesets, have been developed for Asian languages.

C.2 How Locale Affects Processing and Display of Data

As previously mentioned, the locale specified on your system influences how information is processed and displayed. Specifically, locale affects how the software:

- Collates (sorts) data
- Formats date and time expressions
- Formats monetary and other numeric expressions
- Displays messages
- Prompts for yes/no responses

The following sections describe the items in this list.

C.2.1 Collation

Collation is the action of arranging elements of a set into a particular order. Collation always follows a set of rules. Some languages require collation rules that are not used in English.

- Multilevel

Some languages include groups of characters that all sort to the same primary location. Additional sort rules apply to order characters within the same group. For example, the French characters a, á, à, and â all sort to the same primary location. Words that begin with these characters collate the same location, at which point words are sorted within the group. These words are in correct French order:

a

á

abord

âpre

après

âpreté

azur

- One-to-two character mapping

In some languages, certain single characters are treated as if they were two characters. For example, the German sharp s (ß) is sorted as if it were “ss”.

- Multiple-to-one character mapping

Some languages treat a string of characters as if it were a single element. For example, the Spanish ch and ll sequences are treated as unique characters in the Spanish alphabet. The following words are in correct Spanish order:

canto

construir

curioso

chapa

chocolate

dama

- Ignored characters

Some collation rules ignore certain characters. For example, if the hyphen (-) is defined as a character to be ignored, the strings “re-locate” and “relocate” sort to the same position.

Note

This means that you cannot assume that the range [A to z, a to z] includes every letter of an alphabet. For example, the Danish alphabet includes three characters that sort after z.

C.2.2 Date and Time Formats

Users around the world express dates and times with different formatting conventions. When specifying day and month names, people in the United States generally express dates with an expression like the following one:

Tuesday, May 22, 1996

The French, on the other hand, express dates this way:

mardi, 22 mai 1996

The following examples show alternative formats for the date, March 20, 1996. A given format is not the only way to write the date in the listed country:

3/20/96 (United States)

20/3/96 (Great Britain)

20.3.96 (France and Germany)

20-III-96 (Italy)

96/3/20 (Japan)

2/3/20 (Japan, Emperor format)

In Japan's Emperor format, the year (2, in the preceding example) is expressed as the number of years that the current emperor has reigned.

As with dates, there are many conventions for expressing the time of day. In the United States, people often use the 12-hour clock with its a.m. and p.m. designations. People in most other countries use the 24-hour clock to express the time.

In addition to the 12-hour/24-hour clock differences, punctuation for written times can vary, for example:

3:20 p.m. (United States)

15h20 (France)

15.20 (Germany)

15:20 (Japan)

C.2.3 Numeric and Monetary Formats

The characters used to format numeric and monetary values vary from place to place. In the United States, the convention is to use a period (.) as the radix character (the character that separates whole and fractional quantities), and a comma (,) as the thousands separator. In many European countries, these conventions are reversed. For example:

1,234.56 (United States)

1.234,56 (France)

Here are some sample formats for monetary items:

\$1,234.56 (United States, dollars)

kr1.234,56 (Norway, krona)

SFr1.234.56 (Switzerland, Swiss francs)

Note that some formats for monetary amounts include more than two places for fractional digits.

C.2.4 Messages

Programs are sometimes written with English messages embedded in the program itself. In an internationalized program, messages are kept in a separate file and replaced in the program with calls to a messaging system. Messages kept in a separate file can be translated and made available to the program. When translated messages are available, users can interact with the system in their native language.

C.2.5 Yes/No Prompts

Many programs ask questions that need a positive or negative response. Those programs typically look for the English string literals `y` or `yes`, `n` or `no`. An internationalized program lets users enter the characters or words that are appropriate to their language. For example, a French user should be able to enter `o` or `oui`.

C.3 Determining Whether a Locale Has Been Set

If your system is functioning in accordance with the language and conventions of your country, you can assume that the locale has been set correctly.

If you are not sure whether or not your locale has been set, enter the `locale` command to display current settings of the locale environment variables, for example:

```
% locale
LANG=fr_FR.ISO8859-1
LC_COLLATE="fr_FR.ISO8859-1"
LC_CTYPE="fr_FR.ISO8859-1"
LC_MONETARY="fr_FR.ISO8859-1"
LC_NUMERIC="fr_FR.ISO8859-1"
LC_TIME="fr_FR.ISO8859-1"
LC_MESSAGES="fr_FR.ISO8859-1"
LC_ALL=
```

The locale environment variables, described in Section C.4.1, define the locale names used for messages, collation, codeset, numeric formats, monetary formats, date and time formats, and yes/no responses:

```
LANG
LC_COLLATE
LC_CTYPE
LC_NUMERIC
LC_MONETARY
LC_TIME
LC_MESSAGES
LC_ALL
```

In most cases, only the `LANG` variable has been set to a locale name, which then applies to other locale variables with the exception of `LC_ALL`.

C.4 Setting a Locale

When you specify a locale, you specify a locale name that indicates language, territory, and codeset. On Digital UNIX systems, locale names adhere to the following format:

lang_terr.codeset

lang

Is a 2-letter, lowercase abbreviation for the language name. The abbreviations are specified in *ISO 639 Code for the Representation of Names of Languages*, for example: `en` (English), `fr` (French), `de` (German, from “Deutsch”), `ja` (Japanese).

terr

Is a 2-letter, uppercase abbreviation for the territory name. The abbreviations are specified in *ISO 3116 Codes for the Representation of*

Names of Countries, for example: US (United States), NL (the Netherlands), FR (France), DE (Germany, from “Deutschland”), JP (Japan).

codeset

Is a string that identifies the codeset, for example: ISO8859-1 (ISO 8859/1), SJIS (Shift Japanese Industrial Standard), AJEC (Advanced Japanese EUC).

Full locale names include: en_US.ISO8859-1 (English, incorporating customs for the United States), fr_FR.ISO8859-1 (French, incorporating customs for France), de_DE.ISO8859-1 (German, incorporating customs for Germany).

A locale can be set by the system administrator or an individual user. If your system administrator sets the locale at your site, it is likely that a default locale has been specified for all systems, including yours. You can override the default locale if you want to do that.

To set a locale, you assign a locale name to one or more environment variables. The easiest way to do this is to assign a locale name to the LANG environment variable because this variable covers all the pieces of a locale (codeset, collating sequence, numeric, monetary, and date and time formats, messages, and so forth).

Table C-1 lists the locales available when you install the subset, Single-byte European Locales. Additional locales may be available if language-variant software for the operating system is installed on your system.

Table C-1: Locale Names

Language	Country	Codeset	Locale Name
–	–	ASCII	C
–	–	ASCII	POSIX
Danish	Denmark	Latin-1	da_DK.ISO8859-1
German	Switzerland	Latin-1	de_CH.ISO8859-1
German	Germany	Latin-1	de_DE.ISO8859-1
Greek	Greece	Latin-7	el_GR.ISO8859-7
English	Great Britain	Latin-1	en_GB.ISO8859-1
English	United States	Latin-1	en_US.ISO8859-1
Spanish	Spain	Latin-1	es_ES.ISO8859-1
Finnish	Finland	Latin-1	fi_FI.ISO8859-1
French	Belgium	Latin-1	fr_BE.ISO8859-1
French	Canada	Latin-1	fr_CA.ISO8859-1

Table C-1: (continued)

Language	Country	Codeset	Locale Name
French	Switzerland	Latin-1	fr_CH.ISO8859-1
French	France	Latin-1	fr_FR.ISO8859-1
Italian	Italy	Latin-1	it_IT.ISO8859-1
Dutch	Belgium	Latin-1	nl_BE.ISO8859-1
Dutch	The Netherlands	Latin-1	nl_NL.ISO8859-1
Norwegian	Norway	Latin-1	no_NO.ISO8859-1
Portuguese	Portugal	Latin-1	pt_PT.ISO8859-1
Swedish	Sweden	Latin-1	sv_SE.ISO8859-1
Turkish	Turkey	Latin-9	tr_TR.ISO8859-9

The C locale is the default if no locales are set on your system. The POSIX locale is equivalent to the C locale; only letters in the English alphabet are included in the ASCII codeset that is specified for the POSIX and C locales.

C.4.1 Locale Categories

Table C-2 describes environment variables that influence locale functions.

Table C-2: Environment Variables That Influence Locale Functions

Variable	Description
LC_COLLATE	Specifies the collating sequence to use when sorting strings and when character ranges occur in patterns.
LC_CTYPE	Specifies the character classification (codeset) information.
LC_MONETARY	Specifies monetary formats.
LC_NUMERIC	Specifies numeric formats.
LC_MESSAGES	Specifies the language in which messages will appear if translations are available. In addition, this variable specifies strings for affirmative and negative responses.
LC_TIME	Specifies date and time formats.

Table C-2: (continued)

Variable	Description
LC_ALL	Overrides all preceding variables and the LANG environment variable. In general, this variable is used only in programs and should not be set by system managers and users. See the following section on limitations of locale variables for more information.

As is true for the LANG variable, all of the variables in Table C-2 can be assigned locale names. Consider the case where your company is located in the United States but the prevalent language spoken by employees is Spanish. The LANG environment variable could be set to the name of a Spanish language locale and the LC_NUMERIC and LC_MONETARY variables set to the name of a United States English locale. The explicit setting of the LC_NUMERIC and LC_MONETARY variables overrides what they were implicitly set to by LANG. The LC_CTYPE, LC_MESSAGES, LC_TIME, and LC_COLLATE variables would still be implicitly set to the Spanish locale. The following are the variable assignments for the C shell to implement this example:

```
setenv LANG es_ES.ISO8859-1
setenv LC_NUMERIC en_US.ISO8859-1
setenv LC_MONETARY en_US.ISO8859-1
```

The following are the same variable assignments for the Bourne and Korn shells:

```
LANG=es_ES.ISO8859-1
export LANG
LC_NUMERIC=en_US.ISO8859-1
export LC_NUMERIC
LC_MONETARY=en_US.ISO8859-1
export LC_MONETARY
```

Sometimes different versions of the same locale are available locally to meet the needs of certain languages or software applications. The names of such locales end with the at sign (@) plus a modifier field. For example, the collating sequence used for the telephone book in some languages is different from the collating sequence used for dictionaries. If the standard locale for a language defined the dictionary collating sequence, another version of the locale might exist to support the telephone book collating sequence. In this case the alternative locale version might have a name like en_FR.ISO8859-1@phone.

C.4.2 Limitations of Locale Settings

The ability to set locale allows you to tailor your environment, but it does not protect you from making mistakes. The following sections discuss problems that can arise when you define locale variables.

C.4.2.1 Locale Settings Are Not Validated

There is nothing to prevent you from defining implausible combinations of locale names for different aspects of a locale. For example, you could set the `LANG` environment variable to a French locale and the `LC_CTYPE` variable to a Norwegian locale. The results would probably be undesirable; for example, French message translations would likely contain characters not specified in the Norwegian locale. If you define locale variables in addition to `LANG`, you are responsible for ensuring a valid combination of locale settings.

C.4.2.2 File Data Is Not Bound to a Locale

The system has no way of knowing what locale was set when a file was created. Therefore, the system cannot prevent you from processing the file's data using a different locale. For example, suppose you copy to your system a file that was created when the `LANG` variable was set to a German locale. If, on your system, `LANG` is set to a French locale and you use the `grep` command to search for a string in the file, the `grep` command will use French collation and pattern matching rules on the German data. It is therefore your responsibility to know what kind of language data a file contains and to set the locale accordingly.

C.4.2.3 Setting `LC_ALL` Overrides All Other Locale Variables

The `LC_ALL` variable overrides all other locale-dependent environment variables, even if you set it before setting category-specific variables, such as `LC_COLLATE`. The only way to cancel the influence of `LC_ALL` is to undefine the variable. For example, enter the command `unset env LC_ALL`.

The `LC_ALL` variable is available for users familiar with the System V environment. In that environment, users set locale either by setting `LC_ALL` or by setting all the locale category variables individually.

Customizing Your mailx Session

D

You can customize your mailx session permanently by including in your .mailrc file any of the settings described in Table D-1. See the unset command in Appendix F for information about temporary settings.

Table D-1: Variables for Customizing Your mailx Session

Variable	Type	Description
allnet	Binary	All network names with the same login name are treated the same.
append	Binary	Saves messages in your mbox file in the order of arrival; the earliest message is the first message in the file. When this variable is unset, messages are saved in reverse order; the first message in the file is the most recent. The mailx program runs faster if append is set.
ask	Binary	Prompts you for a subject line when you send a message. Enter a blank line to send a message with no subject.
askcc	Binary	Prompts you for carbon-copy recipients for each message you send.
autoprint	Binary	Automatically displays the next message when you delete the current message. When autoprint is unset, mailx does not display the next message when you delete a message. In either case, the next message becomes your new current message.
bang	String	Enables the special-case treatment of the exclamation point (!) in escape command lines as in vi.
cmd	String	Allows the user to specify the default command to be used when using the vertical bar or pipe () command.
conv	String	Allows the user to specify how to convert UUCP style addresses for sendmail.

Table D-1: (continued)

Variable	Type	Description
<code>crt</code>	Numeric	For use with a video display (CRT) terminal. Reads your mail one screenful at a time using the <code>more</code> program. The value tells <code>mailx</code> how many lines of the message to display before invoking the pager. For example: <pre>set crt=20</pre>
<code>DEAD</code>	String	Allows the user to specify a different location for <code>dead.letter</code> . A dead letter will be written to <code>\$HOME/dead.letter</code> by default.
<code>debug</code>	Binary	Displays debugging information.
<code>dot</code>	Binary	Interprets a period on a line by itself to be the end of a message. Do not unset <code>dot</code> and also set <code>ignoreeof</code> .
<code>EDITOR</code>	String	Specifies the pathname for the text editor to be used when you use the <code>edit</code> command or the <code>~e</code> escape. For example: <pre>set EDITOR=/usr/ucb/ex</pre> <p>If your terminal is a CRT terminal, you can specify a screen editor for this variable. See the <code>VISUAL</code> variable later in this table.</p>
<code>escape</code>	String	Allows you to specify the escape character (the character that starts an escape command when you are in the middle of writing a message). The default is the tilde (<code>~</code>). You must specify a single character.
<code>excode</code>	String	Allows the user to specify the locale to be used when doing character conversion on outgoing mail messages.
<code>folder</code>	String	Specifies the directory for storing mail folders. A name beginning with a slash, such as <code>/usr/users/hale</code> , is an absolute pathname. A name without an initial slash is a pathname relative to your home directory. For example, the command <pre>set folder=folder</pre> indicates the directory <code>/usr/users/hale/folder</code> .
<code>header</code>	Binary	Prints the message header of messages when <code>mailx</code> is invoked.
<code>hold</code>	Binary	Prevents messages from being moved to your <code>mbox</code> file after you read them. Messages you have read are held in your system mailbox.

Table D-1: (continued)

Variable	Type	Description
<code>ignore</code>	Binary	Ignores Ctrl/c interrupts, echoing them as “at” signs (@). Note that this variable is different from the <code>ignore</code> command described in Appendix F.
<code>ignoreeof</code>	Binary	Ignores Ctrl/d as the end of an outgoing message. Do not set <code>ignoreeof</code> and also unset <code>dot</code> .
<code>indentprefix</code>	String	Allows the user to specify a string to be inserted at the beginning of each line of text of a mail message that was included using the <code>~m</code> command.
<code>keep</code>	Binary	Allows <code>mailx</code> to truncate your system mailbox instead of deleting it when it is empty. This is useful if you have set special permissions on your system mailbox for security reasons. If <code>keep</code> is unset, your system mailbox is deleted when it becomes empty; the next time it is created, you must reestablish your desired permissions.
<code>keepsave</code>	Binary	Prevents deletion of saved messages when you quit mail. Normally, the <code>mailx</code> program marks messages when you save them in other files or folders, and then deletes them from your system mailbox when you leave <code>mailx</code> . Setting <code>keepsave</code> makes <code>mailx</code> leave these messages in your system mailbox.
<code>lang</code>	String	Allows the user to specify the locale to be used for displaying the mail message.
<code>LISTER</code>	String	Allows the user to specify the command used by the <code>folders</code> commands.
<code>MBOX</code>	String	Allows the user to specify the location for the <code>mbox</code> folder. The <code>mbox</code> folder will normally be located in <code>\$HOME/mbox</code> .
<code>metoo</code>	Binary	Includes you in the list of recipients when you send mail to an alias of which you are a member. If <code>metoo</code> is unset, you will not receive copies of messages sent to aliases of which you are a member.
<code>noheader</code>	Binary	Inhibits display of the header and version identification when you invoke <code>mailx</code> .
<code>nosave</code>	Binary	Prevents <code>mailx</code> from saving aborted messages as <code>dead.letter</code> in your home directory.

Table D-1: (continued)

Variable	Type	Description
onehop	Binary	When responding to a message which contains other recipients, sometimes the addresses of the recipients are relative to the originator's address. The <code>onehop</code> option forces the delivery to not follow the path by which the message arrived and deliver it directly, thereby improving performance.
outfolder	Binary	Causes <code>mailx</code> to save outgoing mail messages in the directory specified in <code>folder</code> .
page	Binary	Causes a form feed to be inserted between messages that are processed by the pipe (<code> </code>) command.
PAGER	String	Allows the user to specify the paging program of their choice to be used when displaying their messages. For example: <code>PAGER=/usr/bin/more</code> or <code>PAGER=/usr/bin/pg</code>
prompt	String	Allows the user to change the <code>mailx</code> prompt when <code>mailx</code> is invoked. For example: <code>prompt=>>></code>
quiet	Binary	Supresses printing the version when first invoked and the message number when you use the <code>type</code> command.
record	String	Specifies the name of a file into which <code>mailx</code> will save copies of all outgoing messages.
Replayall	Binary	Reverses the function of the <code>reply</code> and <code>Reply</code> commands.
save	Binary	Allows the user to save mail messages into <code>dead.letter</code> .
sendwait	Binary	Causes <code>mailx</code> to wait until the message has been processed by the mailer. This option can cause some performance degradation from the users point of view since the user will have to wait until the message has been delivered.
SHELL	String	Allows the user to specify the shell to use when invoking the <code>~</code> or <code>~!</code> commands.
screen	Numeric	Specifies the number of messages to be displayed in one screenful when you enter the <code>headers</code> command.

Table D-1: (continued)

Variable	Type	Description
sendmail	String	Specifies the pathname of the program to use to send mail messages. If this variable is not specified, mailx uses the default delivery system. See your system administrator for information about alternate delivery systems.
showto	Binary	Displays the recipient's name instead of the author's name in message headers.
sign	String	Allows the user to specify a string to be inserted in the mail message when using the ~a command.
Sign	String	Allows the user to specify a string to be inserted in the mail message when using the ~A command.
toplines	Numeric	Specifies the number of lines the top command prints; the default is 5.
verbose	Binary	Invokes top in verbose mode; mailx then announces expansion of aliases as messages are sent.
VISUAL	String	Specifies the pathname for the screen editor that will be used when you use the visual command or the ~v escape. For example: set VISUAL=/usr/ucb/vi If your only terminal is a CRT, you can specify a screen editor for the EDITOR variable, too; then either edit (~e) or visual (~v) will invoke the same editor.

The following example shows the use of the verbose variable, discussed in the previous table, that causes mailx to display expansion of aliases as messages are sent:

```
& set verbose
& mail eve
Subject: Meeting this afternoon
Enter message. Use Ctrl/d to terminate the letter.
Just a reminder, we're meeting at 2.
Ctrl/d
Cc:
/usr/users/debra/.forward: line 0: debra...
forwarding to debra@orange
debra... Connecting to .local...
debra... Sent
```


Using Escape Commands in Your mailx Session

E

There is a special set of commands, called **escape commands** or **escapes**, that perform functions while you are in the process of writing a message.

You use an escape command by entering it on a line by itself, with a tilde (~) as the very first character. The tilde is called an escape character because it signals mailx that an escape command follows. If you want to type a real tilde as the very first character on a line in your message, you must type two tildes.

Table E-1 describes the escape commands.

Table E-1: Escape Commands in mail

Command	Description
~~	Allows the user to enter the tilde (~) character in the body of the mail message.
~! <i>command</i>	Executes the shell <i>command</i> you enter.
~?	Prints a brief summary of escape commands.
~: <i>command</i>	Executes the specified mail command. This is useful for performing housekeeping tasks such as redisplaying a message. For example, entering ~:10 selects and displays message number 10 just as if you had entered its number at the & mail prompt.
~_ <i>command</i>	
~a	Inserts the string set in sign into the mail message.
~A	Inserts the string set in Sign into the mail message.
~b <i>address_list</i>	Inserts the specified names in <i>address_list</i> into the Bcc: (blind carbon copy) list.
~c <i>address_list</i>	Adds the specified names to the Cc: (carbon copy) list.
~C	Dump core.
~d	Includes the file named <code>dead.letter</code> , in your home directory, into the message.
~e	Invokes the editor specified by the EDITOR mail variable to edit the message.

Table E-1: (continued)

Command	Description
<code>~f [message_list]</code>	Reads the current message or the specified messages into your message.
<code>~F [message_list]</code>	Similar to <code>~f</code> with the difference that all headers will be included regardless of any discard, ignore, or retain commands.
<code>~h</code>	Edits the message header fields. This command displays the fields one at a time so you can alter them by adding text to the end, by using the Delete key, or by pressing Ctrl/u to erase the entire field and then retyping it. Use this command with caution.
<code>~i string</code>	Insert the value of the named variable into the mail message. For example: <code>~a</code> is equivalent to <code>~i sign</code> .
<code>~m [message_list]</code>	Includes the current message or the specified messages, shifted one tab stop to the right. This is useful to set off messages you are forwarding as part of your new message.
<code>~M [message_list]</code>	Similar to <code>~m</code> with the difference that all headers will be included regardless of any discard, ignore, or retain commands.
<code>~p</code>	Displays the message you are composing on your terminal. This is useful to see that the message looks the way you want it to and that it includes the right subject heading and lists of recipients.
<code>~r file</code> <code>~< file</code> <code>~<!shell_cmd</code>	Reads the named file into the mail message. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary system command and is executed with the standard output inserted into the mail message.
<code>~q</code> <code>~Q</code>	Aborts the current message as if you press two Ctrl/C interrupts.
<code>~r file</code>	Includes the named file in your message.
<code>~s subject</code>	Makes <i>subject</i> the new subject heading, replacing the previous heading.
<code>~t name...</code>	Adds the names to the To: list of your message.
<code>~v</code>	Invokes the editor specified by the VISUAL mail variable to edit the message.
<code>~w file</code>	Writes the message to the named file.

Table E-1: (continued)

Command	Description
<code>~ command</code> <code>~^</code>	<p>Pipes the message through the named command. This is useful to make global changes in the message; for example, if you are including a message in your new message you can use the <code>sed</code> editor to prefix each line with an angle bracket and a space by using the following command:</p> <pre>~ sed 's/^/> /'</pre> <p>You can then add your own text; the result will look like this:</p> <pre>> This is the text of the message > you have included. > This is the text you add yourself.</pre>

Using the mailx Commands

F

The `mailx` program has a large set of commands, some of which are described in Appendix D and Appendix E. The commands in Table F-1 can help you to use the `mailx` environment more effectively. The `mailx(1)` reference page lists some other commands that are useful only under special circumstances.

Table F-1: Commands for the mailx Program

Command	Description
<code>=</code>	Echoes the number of the current message.
<code>#</code>	Allows the user to write comments in mail script files.
<code>!command</code>	Executes the shell command you enter.
<code>-[n]</code>	Selects and displays the previous message or the <i>n</i> th previous message. For example, <code>-4</code> backs up four messages.
<code>alias</code>	With no arguments, lists the current aliases.
<code>alias alias</code>	With one argument, displays only that alias.
<code>alias alias name...</code>	With two or more arguments, creates an alias with the first argument as its name and all subsequent arguments as the members of the alias. The <code>group</code> command is an alternate for <code>alias</code> .
<code>group</code>	
<code>alternates</code>	Informs <code>mailx</code> that the addresses listed in <code>alternate_list</code> refer to the user. If no <code>alternate_list</code> is specified in the command, the command displays the current list of alternates.
<code>[alternate_list]</code>	
<code>chdir path</code>	Changes your current directory to the pathname specified, as if you had executed the <code>cd</code> shell command except that the directory you specify with <code>chdir</code> prevails only while you are in the mail environment.
<code>cd path</code>	

Table F-1: (continued)

Command	Description
<code>copy [message_list] file</code> <code>co [message_list] file</code>	Copies the current message or the specified messages into a file. If <i>file</i> exists, the messages are appended. This command works like <i>save</i> except that it does not mark copied messages for deletion when you quit from <i>mailx</i> .
<code>Copy [message_list]</code>	Saves the specified messages in a file whose name is derived from the author of the first message in the <i>message_list</i> . This command will not mark the messages as being saved. Otherwise equivalent to the <i>Save</i> command.
<code>delete [message_list]</code> <code>d [message_list]</code>	Deletes the current message or the specified messages. You can use the <i>undelete</i> command to recover messages you have accidentally deleted.
<code>discard [field_list]</code>	Identical to the <i>ignore</i> subcommand.
<code>dp</code> <code>dt</code>	Deletes the current message and prints the next active message.
<code>echo string</code>	Echo the given string. Similar to the shell <i>echo</i> command.
<code>edit [message_list]</code>	Invokes the editor specified by <i>EDITOR</i> and loads <i>message_list</i> into the editor. When you exit, any changes made are saved back into <i>message_list</i> .
<code>exit</code> <code>ex</code> <code>x</code>	Exits mail without updating your system mailbox.
<code>file [file]</code> <code>fi [file]</code> <code>folder [file]</code> <code>fo [file]</code>	Selects a mail file or folder. If you do not specify a file, this command prints your current path and file name and the number of messages in your current file. If you specify a file or folder, this command displays any changes you have made to your current file and switches to the specified file for reading.
<code>folders</code>	Lists the names of the folders in your folder directory.

Table F-1: (continued)

Command	Description
<code>followup message</code>	Responds to a message and record the response in a file whose name is derived from the author of the message. This command overrides the <code>record</code> option if set.
<code>Followup [message_list]</code>	Responds to the first message in <code>message_list</code> and sends the message to the author of each message in <code>message_list</code> . The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message.
<code>from [login] f [login]</code>	Prints the active message header. If you specify a login name, this command prints all the active messages from the specified name.
<code>headers [n] h [n]</code>	Lists active message headers, using the value of the <code>screen</code> variable as the number of headers to display. See Appendix D for a description of the <code>screen</code> variable. If you have more than one screenful of messages, you can move forward or backward one screenful with the <code>z</code> command. If you specify a message number, the <code>headers</code> command displays the screenful that includes the specified message.
<code>help</code>	Displays help information.
<code>hold [message_list] ho [message_list] preserve [message_list] pre [message_list]</code>	Holds, or preserves, the current message or the specified specified messages in your system mailbox instead of moving them to your <code>mbox</code> file.
<code>if condition else endif</code>	Construction for conditional execution of <code>mailx</code> subcommands. Subcommands following <code>if</code> are executed if <code>condition</code> is True. Subcommands following <code>else</code> are executed if <code>condition</code> is not True. An <code>else</code> is not required but the <code>endif</code> is required. The condition can be <code>send</code> for sending mail, or <code>receive</code> for receiving mail.

Table F-1: (continued)

Command	Description
<code>ignore [field...]</code>	Sets <code>mailx</code> to display messages without the specified fields of the header when you use the <code>print</code> or <code>type</code> command. Note that this command is different from the <code>ignore</code> variable described in Appendix D. If you enter the <code>ignore</code> command with no arguments, the current list of ignored fields is displayed.
<code>list</code>	Displays a list of valid <code>mailx</code> subcommands.
<code>local</code>	List other names for the local host.
<code>mail user_name</code> <code>m user_name</code>	Sends a message to the specified user.
<code>mbox [message_list]</code>	Marks the current message or the specified messages to be moved to your <code>mbox</code> file. This is helpful if you have set the <code>hold</code> variable in your <code>.mailrc</code> file.
<code>more [message_list]</code>	Displays the messages in <code>message_list</code> using the defined pager program in <code>PAGER</code> . Identical to the <code>page</code> subcommand.
<code>More [message_list]</code>	Similar to the <code>more</code> subcommand, but also displays the ignored header fields. See <code>more</code> and <code>ignore</code> subcommands.
<code>new [message_list]</code>	Marks each message in the <code>message_list</code> as not having been read. Identical to <code>New</code> , <code>unread</code> , and <code>Unread</code> subcommands.
<code>New [message_list]</code>	Marks each message in the <code>message_list</code> as not having been read. Identical to <code>new</code> , <code>unread</code> , and <code>Unread</code> subcommands.
<code>page [message_list]</code>	Displays the messages in <code>message_list</code> using the defined pager program in <code>PAGER</code> . Identical to the <code>more</code> subcommand.
<code>Page [message_list]</code>	Similar to <code>page</code> but also displays the ignored header fields. Identical to the <code>More</code> subcommand.

Table F-1: (continued)

Command	Description
<code>pipe [message_list] [shell_command] [message_list] [shell_command]</code>	Pipes the <i>message_list</i> through the <i>shell_command</i> . The message is treated as being read. If no arguments are given, the current message is piped through the command given in <i>cmd</i> . If the <i>page</i> option is set, a formfeed is inserted after each message.
<code>next n + Return</code>	Displays the next message.
<code>Print [message] P [message] Type [message] T [message]</code>	Displays the current message or the specified message, including any header fields specified by the <i>ignore</i> command.
<code>print [message] p [message] type [message] t [message]</code>	Displays the current message or the specified message without any header fields specified by the <i>ignore</i> command.
<code>quit q</code>	Leaves the <i>mailx</i> program and updates your system mailbox. If you do not have the <i>hold</i> variable set, all messages that you have not deleted, saved, or preserved are moved to your <i>mbox</i> file. If you do have <i>hold</i> set, all these messages will be left in your system mailbox and marked as having been read.
<code>Reply R Respond</code>	Replies to a message. If the original message was addressed to a group of people, replies sent with the <i>Reply</i> command are sent only to the originator of the message.
<code>reply r respond</code>	Replies to a message. If the original message was addressed to a group of people, replies sent with the <i>reply</i> and <i>respond</i> commands are sent to everyone who received the original message.
<code>retain [field_list]</code>	Adds the header fields in <i>field_list</i> to the list of headers to be retained when displaying message with the <i>print</i> or <i>type</i> subcommands. Use <i>type</i> and <i>print</i> to view messages in their entirety, including fields that are not retained. If <i>retain</i> is executed with no arguments, it lists the current set of retained fields.

Table F-1: (continued)

Command	Description
<code>save [message_list] file</code>	Saves the current message or the specified messages in the file. Note that the messages are added to the specified file so that you will not delete the contents of the file.
<code>s [message_list] file</code>	Saves the specified messages in a file whose name is derived from the author of the first messages. The name of the file is assumed to be the author's name with all network addressing stripped off.
<code>Save [message_list]</code>	Saves the specified messages in a file whose name is derived from the author of the first messages. The name of the file is assumed to be the author's name with all network addressing stripped off.
<code>set [variable] se [variable]</code>	If entered with no variables, the <code>set</code> command displays all the options you have set. If you specify a variable, the option will be set. (Appendix E lists the available variables.)
<code>shell sh</code>	Invokes the shell interactively.
<code>source file so file</code>	Reads mail commands from a file (usually <code>.mailrc</code>).
<code>size [message_list]</code>	Displays the size in lines and characters of the messages in <code>message_list</code> .
<code>top [message_list] to [message_list]</code>	Displays the first five lines in the current message or each of the specified messages.
<code>touch [message_list]</code>	Marks the messages in <code>message_list</code> to be moved from your system mailbox to your personal mbox when you quit the <code>mailx</code> program even though you have not read the listed messages. The messages appear in your mbox as unread messages. When you use <code>touch</code> , the last message in <code>message_list</code> becomes the current message.
<code>unalias alias_list</code>	Deletes the specified alias names.
<code>undelete message_list u message_list</code>	Undeletes the specified messages.
<code>unread [message_list]</code>	Marks each message in <code>message_list</code> as not having been read. Identical to the <code>new</code> , <code>New</code> , and <code>Unread</code> subcommands.
<code>Unread [message_list]</code>	Marks each message in <code>message_list</code> as not having been read. Identical to the <code>new</code> , <code>New</code> , and <code>unread</code> subcommands.

Table F-1: (continued)

Command	Description
<code>unset</code>	Unsets (turns off) options. For example, if your <code>.mailrc</code> file includes a <code>set hold</code> command, you can use the <code>unset</code> command to disable the <code>hold</code> variable for the current <code>mailx</code> session.
<code>version</code>	Displays the version banner for the <code>mailx</code> command.
<code>visual</code>	Invokes the editor specified by the <code>VISUAL</code> mail variable to edit the current message.
<code>write</code> <code>[message_list] file</code> <code>w [message_list]</code> <code>file</code>	Saves the current message or the specified messages in the named file. This is similar to the <code>save</code> command, except that <code>write</code> saves only the body of each message; the headers are deleted.
<code>z[+]</code> <code>z-</code>	Moves forward or backward one screenful of messages. You can specify the number of messages in a screenful with the <code>screen</code> variable. (See Appendix D.) To move forward one full screen, enter <code>z</code> or <code>z+</code> ; to move backward, enter <code>z-</code> .

Index

Special Characters

! subcommand (ftp), 12–8

\$, 1–2

%, 1–2

& operator, 6–7

&& operator, 7–8

() (parentheses)

See parentheses

; (semicolon)

See semicolon

? (question mark)

See question mark

? subcommand (ftp), 12–8

{ } (braces)

See braces

|| operator, 7–8

~ (tilde)

See tilde

A

absolute pathname, 2–10

absolute permissions, 5–11

removing, 5–11

accessing

help, 1–9

account subcommand (ftp), 12–5

active processes, 6–16

add (a) command (vi text editor), A–10

ali command in MH, 11–20

alias

listing in MH, 11–20

aliases, 8–10, 8–30

C shell, 8–10, 8–13

for mail, 11–16

Korn shell, 8–34

seeing current when in mail, 11–16

setting in C and Korn shells, 7–3

ampersand (&) operator

background processes, 6–7

anno command in MH, 11–20

append subcommand (ed editor), B–2

append text (A) command (vi text editor),

A–11

apropos command, 1–10

arguments to commands, 1–5

ASCII

codeset, C–2

ascii subcommand (ftp), 12–5

Asian languages

codesets that support, C–2

ask variable in mail, 11–16

askcc variable in mail, 11–17

assign default values to shell, 7–13

at (@) extension in locale name, C-9
autologin, 1-1

B

background

putting process in, 6-14

background process, 6-6, 6-7

displaying status for, 6-10

backslash, 7-12

to continue a command on the next line,
11-16

bg command, 6-14

bg command (C shell), 8-13

binary numbers

in permissions, 5-12

binary subcommand (ftp), 12-5

Bourne shell, 7-1, 8-1, 8-14

built-in commands, 8-18

built-in variables, 8-17

clearing variable values, 7-23

login script, 8-14

.logout script, 7-25

metacharacters, 8-16

pattern matching, 2-14

.profile login script, 7-17, 8-14

redirecting errors, 6-4

setting variables, 7-19

braces

using to group commands, 7-11

breaking remote cu connection (UUCP), 14-8

breaking remote tip connection (UUCP),

14-14

built-in

commands, 8-12, 8-18, 8-33

variables, 8-11, 8-17, 8-32

burst command in MH, 11-20

bye subcommand (ftp), 12-5

C

C locale, C-7

C shell, 7-1, 8-1, 8-3

aliases, 7-3, 8-10

built-in commands, 8-12

built-in variables, 8-11

changing to another shell, 7-6

clearing variable values, 7-23

command history, 7-3, 8-8

.cshrc login script, 7-17, 8-3

displaying value of variables, 7-22

filename completion, 7-3, 8-9

.login script, 7-17, 8-5

.logout script, 7-25

metacharacters, 8-6

redirecting errors, 6-5

setting environment variables, 7-20

stop and start a process, 6-13

can't remember command name, 1-11

cancel print jobs, 3-13

canceling commands, 1-6

carbon copies in mail, 11-3

getting a prompt for, 11-17

case sensitivity, 2-6

cat command, 3-6

cd command, 3-1, 4-4

Bourne shell, 8-19

Korn shell, 8-34

cd subcommand (ftp), 12-7

cdup subcommand (ftp), 12-7

change (c) command (vi text editor), A-14

change (c) subcommand (ed editor), B-20

change word (cw) command (vi text editor),

A-14

changing

- directories, 3-1, 4-4
- directory permissions, 5-10
- file permissions, 5-10
- group, 5-19
- identity, 5-17
- name of file during copy command, 3-22
- owners of files and directories, 5-19
- shell permanently, 7-6
- shell temporarily, 7-5
- your password, 1-8
- your shell, 7-4

character mapping

- multiple-to-one, C-3
- one-to-two, C-3

characters

- list of pattern matching, 2-14
- maximum number in filename, 2-7
- quoting to make literal, 7-11
- upper and lower case, 2-6

chgrp

- command, 5-4

chgrp command, 5-19, 5-20

chmod command, 5-13, 5-3, 5-4, 5-8

chown command, 5-19, 5-20

clearing variable values, 7-22

close subcommand (telnet), 13-4

codesets

- ASCII, C-2
- eight-bit, C-2
- ISO, C-2
- part of locale, C-2
- support for Asian languages in, C-2

collation, C-3

colon

- use in vi text editor, 2-3, A-3, A-18

command argument

- containing more than one word, 1-11

command history, 7-3

- C shell, 8-8
- Korn shell, 8-25

command mode (vi editor), A-6

commands

- alias, 8-13, 8-34
- apropos, 1-10
- bg, 6-14, 8-13
- can't remember command name, 1-11
- cat, 3-6
- cd, 3-1, 4-4, 8-19, 8-34
- chgrp, 5-19, 5-4
- chmod, 5-4, 5-8
- chown, 5-19
- connecting with pipes, 7-8
- continuing on the next line, 11-16
- cp, 3-19, 3-20
- date, 1-5
- df, 3-14
- diff, 3-24
- documentation for, 1-9
- echo, 7-22, 8-13, 8-19, 8-34
- exit, 5-18
- export, 8-19, 8-34
- fc, 8-34
- fg, 6-14, 8-13
- file, 3-30
- find, 6-7
- flags, 1-5
- history, 8-13, 8-34
- in mailx, F-1

commands (cont.)

jobs, 6–11, 6–9, 8–13, 8–34
kill, 6–12
ln, 3–13, 3–15
login, 1–1
logout, 8–13
lpq, 3–12, 3–13
lpr, 3–10, 3–11
lprm (remove from print queue), 3–13
lpstat, 3–13
ls, 2–13, 3–18, 3–2, 3–23, 3–3, 3–4, 5–5,
 5–6
man, 1–10, 1–5, 1–9
mkdir, 3–2, 4–1
more, 3–15, 3–6, 3–8
mv, 3–22, 3–23, 4–10
options, 1–5
page (pg), 3–6
passwd, 1–6, 1–8
pg, 3–6
pr, 3–7
ps, 6–8, 6–9
pwd, 2–7, 8–19, 8–34
redirecting output, 3–27
rehash, 8–13
repeat, 8–13
rm, 3–17, 3–19, 3–27, 3–28, 4–13
rmdir, 4–10, 4–12
run from remote host (UUCP), 14–18
running conditionally, 7–8
running in sequence, 7–7
running multiple, 7–7
set, 8–13, 8–19, 8–34
setenv, 7–20, 8–13
sort, 3–26
source, 8–13

commands (cont.)

stopping execution, 1–6
su, 5–17
time, 8–13
times, 8–19, 8–34
touch, 3–29
trap, 8–19, 8–34
umask, 5–13, 8–19, 8–34
unalias, 8–13, 8–34
unset, 7–23, 8–13, 8–19, 8–34
unsetenv, 7–23, 8–13
use of special characters in, 7–11
using, 1–4
w, 6–15
wc, 6–3
who, 6–14
whoami, 5–17

communicating with remote host (UUCP),

14–1

comp command in MH, 11–20**comparing files, 3–24****comparison between shell features, 8–2****completing filenames, 7–3****computer virus, 5–21****concatenate files, 3–7****conditional running of commands, 7–8****connecting commands with pipes, 7–8****connecting to an unknown remote host via**

modem (UUCP), 14–10

connecting to an unknown remote system via

modem (UUCP), 14–4

context searching

ed editor, B–12

vi text editor, A–16

continuing a command on the next line, 11–16

copying

- changing file name during, 3–22
- files from one directory to another, 3–20
- files to other directories, 3–21
- files (UUCP), 14–22
- files, local host control (UUCP), 14–24
- lines, ed editor, B–23

copying directories, 4–8

copying files, 3–19

correcting mistakes

- in commands, 1–5
- when logging in, 1–2

correcting typing errors

- ed editor, B–3
- vi text editor, A–11

cp command, 3–19

creating

- directories, 3–2, 4–1
- empty files, 3–29
- logout script, 7–24
- mail folders, 11–13
- multiple names for same file, 3–19
- symbolic links, 3–15
- text files
 - ed editor, B–2
 - vi editor, A–4
- text files with vi, 2–2

CRT screen

- use by talk command, 11–24

crt variable in mail, 11–17

csh.login system login script, 7–16

.cshrc login script

- modifying to use MH program, 11–17, 7–17, 8–3
- modifying for System V Habitat, 9–2

ct command (UUCP)

- options, connecting to remote host via modem, 14–14

Ctrl/C

- to abort a mail message, 11–5

Ctrl/D, 5–18, 5–19

- to end a mail message, 11–17
- to log out, 1–4

cu command (UUCP)

- connect local to remote, 14–8
- options, connecting to a remote host, 14–3
- using local commands, 14–3, 14–6

current directory, 2–7

current message, 11–9

customizing

- login scripts, 7–18

customizing mailx, D–1

customizing your mail environment

- by customizing the mail program, 11–15
- by setting mail variables, 11–16

D

database security, 5–1

- group, 5–3

date command, 1–5

date format, C–4

dead.letter file, 11–5

default

- permission codes, 5–13, 5–7
- prompts for shells, 7–5
- setting permissions with umask, 5–13
- values to shells, 7–13

default user mask (umask), 5–17

defining

- custom shell variables, 7–19
- login account, 5–2

defining (cont.)

user environment, 7–13

delete character (x) command (vi text editor),

A–13

delete (d) subcommand (ed editor), B–18

delete key, 1–5

delete line (dd) command (vi text editor),

A–13

delete subcommand (ftp), 12–7

delete word (dw) command (vi text editor),

A–13

deleting

a specific line

ed editor, B–19

vi text editor, A–13

clearing a line (D) command, A–14

current line

ed editor, B–18

directories, 4–10

files, 3–28, 3–29

multiple lines

ed editor, B–19

vi text editor, A–13

one character at a time, A–13

print jobs from queue, 3–13

determining file type, 3–30

device name, specifying with cu command

(UUCP), 14–5

diff command, 3–24

differences between file and directory

permissions, 5–5

differences between files, 3–24

digests

exploding into messages in MH, 11–20

directories, 2–7

changing, 3–1, 4–4

directories (cont.)

changing owner of, 5–19

changing permissions, 5–10

copying, 4–8

copying files from another directory, 3–20

creating, 4–1

definition of, 2–1

deleting, 4–10

displaying, 4–8

displaying current, 2–7

listing contents of, 3–2

managing, 4–1

parent, 2–10

path, 2–10

permissions, 5–5

removing current, 4–12

removing empty, 4–11

removing multiple, 4–12

renaming, 4–10

root, 2–9

search path, 7–23

tree structure, 2–8

disk partitions, 3–14

display log of UUCP utilities, 14–31

display subcommand (telnet), 13–4

displaying

active processes, 6–16

command status, 6–8

current directory name, 2–7

differences between files, 3–24

directories, 4–8

directory contents, 3–2

directory permissions, 5–5

disk partitions, 3–14

file permissions, 5–5

file type, 3–30

displaying (cont.)

- files, 3–5
- files with formatting, 3–7
- files without formatting, 3–6
- inactive users, 6–15
- inode number of file, 3–16
- multiple files at once, 3–6
- online reference pages, 1–9
- pathnames for all files, 6–8
- print queue status, 3–12
- printer names, 3–13
- process information, 6–14
- process status, 6–9
- user identity, 5–18
- user name, 5–17
- value of variables in C shell, 7–22
- variable name, 7–12
- variable values, 7–22
- who is logged on, 1–10
- who is on the system, 6–14

dist command in MH, 11–20

dot notation, 2–11

dot option for mail, 11–17

double quotes, 7–13

E

echo command, 7–22

Bourne shell, 8–19

C shell, 8–13

Korn shell, 8–34

ed editor

- append subcommand, B–2
- change (c) subcommand, B–20
- context searching, B–12
- copying lines, B–23
- correcting typing errors, B–3

ed editor (cont.)

- creating and saving text files, B–2
 - delete (d) subcommand, B–18
 - deleting a specific line, B–19
 - deleting current line, B–18
 - deleting multiple lines, B–19
 - displaying the current line, B–9
 - edit buffer, B–1
 - edit (e) subcommand, B–6, B–7
 - edit (ed) command, B–7
 - global (g) operator, B–15, B–16
 - insert (i) subcommand, B–22
 - locating text, B–12
 - move (m) subcommand, B–19
 - moving text, B–19
 - print (p) subcommand, B–3
 - quit (q) subcommand, B–6
 - read (r) subcommand, B–6, B–8
 - removing characters, B–16
 - replacing character strings, B–13
 - saving part of a file, B–5
 - saving text, B–4, B–5
 - starting the editor, B–2
 - substitute (s) subcommand, B–13
 - substitutions on multiple lines, B–15
 - transfer (t) subcommand, B–23
 - using system commands, B–24
 - write (w) subcommand, B–4, B–6
- edit (e) subcommand (ed editor)**, B–6, B–7
- edit (ed) command (ed editor)**, B–7
- editing**
- command lines in the Korn shell, 8–27
 - linked files, 3–13
 - mail messages, 11–4
- empty files**, 3–29

end of message/conversation (local communications), 11–22, 11–23

ending

- a local message, 11–22
- a mail message, 11–3

enhanced security system, 1–1

environment variables, 7–14

- corresponding to locale categories, C–8
- HOME, 7–15
- LANG, 7–15
- LC_ALL, 7–15
- LC_COLLATE, 7–15
- LC_CTYPE, 7–15
- LC_MESSAGES, 7–15
- LC_MONETARY, 7–15
- LC_NUMERIC, 7–15
- LC_TIME, 7–15
- LOGNAME, 7–15
- MAIL, 7–15
- PATH, 7–15
- SHELL, 7–15
- TERM, 7–15
- TZ, 7–15

erasing files

- prevention, 3–20

errorbells environment variable, A–23

errors

- redirecting output to a file, 6–4

escape character

- including in a mail message, E–1
- tilde as in mail, E–1

escape commands in mail, E–1

escape key

- use in vi, A–3

/etc/group file, 5–3

/etc/passwd file, 5–2, 5–3

/etc/password file, 7–13

ex line editor, A–18

execute permission, 5–5, 5–9

exit command, 5–18, 5–19

exiting from the mail program, 11–15

export command

- Bourne shell, 8–19
- Korn shell, 8–34

F

fc command (Korn shell), 8–34

fg command, 6–14

fg command (C shell), 8–13

file command, 3–30

file descriptors, 6–4

file system, 2–1, 2–5

file type

- determining, 3–30

filename completion, 7–3, 8–29, 8–9

filenames, 2–5

- characters restricted in, 2–6

files

- changing identity to access, 5–17
- changing permissions, 5–10
- comparing differences between two, 3–24
- copying, 3–19
- copying (UUCP), 14–22
- creating empty, 3–29
- definition of, 2–5
- descriptors, 6–4
- determining type, 3–30
- displaying, 3–6
- displaying multiple, 3–6
- displaying pathnames for all, 6–8
- displaying with formatting, 3–7

files (cont.)

- inode number, 3–16
- introduction to, 2–1
- linking, 3–13
- mailing to other users, 11–6
- maximum length, 2–7
- moving, 3–22
- naming conventions, 2–6
- noclobber variable to prevent erasure, 3–20
- printing, 3–10
- protecting, 5–3
- read permission, 5–9
- reading input from, 6–2
- receiving (UUCP), 14–21
- redirecting errors to, 6–4
- redirecting output, 6–3
- removing, 3–27
- renaming, 3–22
- restricted characters in filename, 2–6
- restricting access, 5–15
- saving mail messages in, 11–13
- security, 5–1
- security considerations, 5–20
- sending (UUCP), 14–21
- setting absolute permissions, 5–11
- sorting contents of, 3–26
- specifying with pattern matching, 2–13
- used for security control, 5–2
- wildcard use, 2–13

filtering standard input, 7–9

filters, 7–8

find command, 6–7

finger command, 10–2

flags, 1–5

folder command

- in MH, 11–20

folder command in mail, 11–13

folder variable for mail, 11–17

folders command

- in MH, 11–20

folders in mail, 11–12, 11–20

- creating, 11–13

- listing in MH, 11–18, 11–20

- names of in MH program, 11–18

- removing in MH, 11–20

- seeing what your current folder is, 11–13

- setting up to use, 11–12

- used by MH program, 11–18

foreground

- putting process in, 6–14

foreground processes, 6–6

forget command name, 1–11

formatting a file, 3–7

forw command in MH, 11–20

forwarding files (UUCP), 14–14

forwarding mail messages, 11–14

ftp subcommands, 12–5, 12–7, 12–8

full pathname, 2–10

G

get subcommand (ftp), 12–5

getting help, 1–9

global (g) operator (ed editor), B–15, B–16

global substitution in vi text editor, A–20

group file, 5–1

grouping commands

- with braces, 7–11

- with parentheses, 7–11

groups, 5–3

guidelines

- for setting password, 1–7

H

hard links, 3–14

help, 1–9

getting from the mail program, 11–14

help command in mail, 11–14

help subcommand (ftp), 12–8

history command

C shell, 8–13

Korn shell, 8–34

history of recently used commands, 8–25, 8–8

HOME environment variable, 7–15

I

i-number, 3–16

I/O, 6–1

ignorecase environment variable, A–23

illegal characters in filenames, 2–6

inbox folder in MH program, 11–18

inc command in MH, 11–18, 11–20

including mail in MH, 11–18, 11–20

inline editing in Korn Shell, 7–4

inode number, 3–16

and symbolic links, 3–17

moving files, 3–22

input

reading, 6–2

input mode (vi text editor), A–10

insert (i) subcommand (ed editor), B–22

insert text (I) command (vi text editor), A–11

insert text (i) command (vi text editor), A–10

intermediate hosts used in file transfers

(UUCP), 14–14

internationalization

LANG environment variable, 7–15, C–7

LC_ALL environment variable, 7–15, C–8

internationalization (cont.)

LC_COLLATE environment variable, 7–15,
C–8

LC_CTYPE environment variable, 7–15,
C–8

LC_MESSAGES environment variable,
7–15, C–8

LC_MONETARY environment variable,
7–15, C–8

LC_NUMERIC environment variable, 7–15,
C–8

LC_TIME environment variable, 7–15, C–8
pattern matching, 2–14

ISO codesets, C–2

J

jobs command, 6–11, 6–9

C shell, 8–13

Korn shell, 8–34

K

kill command, 6–12

killing a job or process, 6–12

Korn shell, 7–1, 8–1, 8–19

aliases, 7–3, 8–30

built-in commands, 8–33

built-in variables, 8–32

clearing variable values, 7–23

command history, 7–3, 8–25

editing command lines, 8–27

filename completion, 7–3, 8–29

inline editing, 7–4

.kshrc login script, 7–17, 8–22

login script, 8–20, 8–22

.logout script, 7–25

Korn shell (cont.)

- metacharacters, 8–24
- pattern matching, 2–14
- .profile login script, 7–17, 8–20
- redirecting errors, 6–4
- setting variables, 7–19

.kshrc login script, 7–17, 8–22

L

LANG environment variable, 7–15, C–7

language

- part of locale, C–1

LC_ALL environment variable, 7–15, C–8

- dangers of setting, C–10

LC_COLLATE environment variable, 7–15, C–8

LC_CTYPE environment variable, 7–15, C–8

LC_MESSAGES environment variable, 7–15, C–8

LC_MONETARY environment variable, 7–15, C–8

LC_NUMERIC environment variable, 7–15, C–8

LC_TIME environment variable, 7–15, C–8

lcd subcommand (ftp), 12–7

length

- maximum for files, 2–7

linking files, 3–13, 3–15

links

- hard, 3–14
- removing, 3–17
- soft, 3–14
- symbolic, 3–14

list

- of pattern matching characters, 2–14

listing directory contents, 3–2, 3–3, 3–2

listing mail messages, 11–7

literal characters, 7–11

- using backslash, 7–12
- using double quotes, 7–13
- using single quotes, 7–12

ln command, 3–13, 3–15

local commands (UUCP), 14–12, 14–3, 14–6, 14–9

local communication facility, messages

- sending, write command, 11–21

local host control of file access (UUCP)

- uuto command, 14–24

local variables, 7–16

locale command, C–5

locales

- categories of, C–8
 - determining which locale is set, C–5
 - effect on date and time format, C–4
 - effect on messages, C–5
 - effect on software, C–2
 - effect on sort order, C–3
 - environment variables used with, C–6
 - introduction to, C–1
 - name format for, C–6
 - names of
 - at (@) modifier in, C–9, C–7
 - numeric and monetary format in, C–5
 - setting, C–7
 - preferred method for, C–7
 - restrictions when, C–10
 - yes/no response strings, C–5
- locating command names**, 1–10
- locating text**
- ed editor, B–12
 - vi text editor, A–16

logging in

See login

logging into another user account, 5–17

logging out, 1–4

script, 7–24

login

with enhanced security system, 1–1

login account, 5–2

login as root user, 5–19

login directory, 2–7

.login login script

modifying to use MH program, 11–17

modifying for System V Habitat, 9–2

login program, 7–13

login script

activating umask, 5–16

Bourne shell, 8–14

C shell, 8–3, 8–5

csh.login system script, 7–16

.cshrc script, 7–17, 8–3

Korn shell, 8–20, 8–22

.kshrc script, 7–17, 8–22

.login script, 7–17, 8–5

.profile script, 7–17, 8–14, 8–20

profile system script, 7–16

login scripts

customizing, 7–18

LOGNAME environment variable, 7–15

logout command (C shell), 8–13

logout script, 7–24, 7–25

lowercase characters, 2–6

lpq (display print queue) command, 3–12

lpr command, 3–10

lprm (remove from print queue) command,

3–13

ls command

output from -l option, 3–4

ls (list directory contents) command, 3–2

ls (list directory) command, 3–18, 3–2, 3–3,
3–4, 5–5, 5–6

ls subcommand (ftp), 12–7

M

Mail, 11–1 to 11–17

mail

aliases, 11–16

aliases, listing in MH, 11–20

annotating messages in MH, 11–20

announcement of new messages on arrival,
11–7

checking for messages in MH, 11–20

composing messages in MH, 11–20

compressing messages into a file in MH,
11–20

current message, defined, 11–9

customizing, 11–15

deleting messages in MH, 11–20

displaying messages in MH, 11–20

distributing messages in MH, 11–20

editing a message, 11–4

ending a message, 11–3

entering a subject for, 11–16, 11–3

exiting from, 11–15

exploding digests in MH, 11–20

-f option to select a folder, 11–13

filing messages in MH, 11–20

folders, 11–12

creating, 11–13

setting up to use, 11–12

formatted listings of messages in MH, 11–20

forwarding messages in, 11–14

mail (cont.)

- forwarding messages in MH, 11–20
- handling with the MH message handling program, 11–17
- getting help in, 11–14
- including messages in MH
 - asynchronously, 11–20, 11–18, 11–20
- listing folders in MH, 11–18, 11–20
- listing message headers, 11–7, 11–8
- listing messages in MH, 11–20
- mailing files in, 11–6
- marking messages in MH, 11–20
- messages moved to mbox file, 11–10
- notification when messages are waiting for you, 11–7
- preventing moving of messages to mbox file, 11–10
- reading messages in, 11–7
 - by number, 11–8
- reading messages in MH, 11–20
- removing folders in MH, 11–20
- removing messages in MH, 11–20
- replying to messages in, 11–10
- replying to messages in MH, 11–20
- reporting recipients of messages in MH, 11–20
- saving messages in files, 11–13
- saving messages in folders, 11–13
- seeing what your current folder is, with the folder command, 11–13
- selecting messages by content in MH, 11–20
- sending messages, 11–2, 11–3
 - by carbon copy, 11–17, 11–3
- sending messages in MH, 11–20
 - with a prompting front end, 11–20
- sending messages to aliases, 11–16

mail (cont.)

- setting or listing a folder in MH, 11–20
- sorting messages in MH, 11–20
- specifying length of screen for message display, 11–17
- specifying location of folders for, 11–17
- specifying location of record copies of outgoing mail, 11–17
- specifying options interactively, D–1
- the MH program, 11–17, 11–21
- variables, 11–16 to 11–17
- verbose mode, example of, D–5

MAIL environment variable, 7–15

mailing files, 11–6

- from the shell, 11–6

.mailrc file

- modifying to customize mail, 11–15, 11–12

mailx, 11–1 to 11–17

- escape commands, E–1

mailx command

- to enter the mail environment, 11–7

mailx commands, F–1

man command, 1–5, 1–9

man pages, 1–9

managing directories, 4–1

map command (vi text editor), A–23

mark command in MH, 11–20

maximum length for file names, 2–7

mbox file, 11–10

- preventing moving of messages to, 11–10

message

- annotating in MH, 11–20
- checking for in MH, 11–20
- composing in MH, 11–20
- compressing into files in MH, 11–20
- deleting in MH, 11–20

message (cont.)

- displaying a list of in MH, 11–18
- displaying in MH, 11–20
- distributing in MH, 11–20
- exploding digests into in MH, 11–20
- filing in MH, 11–20
- formatted listings of in MH, 11–20
- forwarding in MH, 11–20
- including in MH
 - asynchronously, 11–20, 11–18, 11–20
- listing in MH, 11–20
- marking in MH, 11–20
- reading in MH, 11–20
- removing in MH, 11–18, 11–20
- replying to in MH, 11–20
- reporting recipients of in MH, 11–20
- selecting by content in MH, 11–20
- sending in MH, 11–20
 - with a prompting front end, 11–20
- sorting in MH, 11–20

messages, C–5

messages (local communications)

- ending, end-of-file symbol (EOF), 11–22,
11–23
- long, in files, 12–9
- sending, write command, 11–21

metacharacters

- Bourne shell, 8–16
- C shell, 8–6
- Korn shell, 8–24

mget subcommand (ftp), 12–5

MH message handling program, 11–17, 11–21

- annotating messages in, 11–20
- checking for messages in, 11–20
- commands used at the shell prompt, 11–17
- composing messages in, 11–20

MH message handling program (cont.)

- compressing messages into a file in, 11–20
- deleting messages in, 11–20
- distributing messages in, 11–20
- exploding digests in, 11–20
- filing messages in, 11–20
- finding if installed on your host, 11–17
- forwarding messages in, 11–20
- including messages in
 - asynchronously, 11–20, 11–20
- listing aliases in, 11–20
- listing folders in, 11–20
- listing messages in, 11–20
- marking messages in, 11–20
- modifying your path to use, 11–17
- reading messages in, 11–18, 11–20
- removing folders in, 11–20
- removing messages in, 11–18, 11–20
- replying to messages in, 11–20
- reporting recipients of messages in, 11–20
- selecting a folder in, 11–18
- selecting messages by content in, 11–20
- sending messages in, 11–20
 - with a prompting front end, 11–20
- setting or listing a folder in, 11–20
- sorting messages in, 11–20
- tailoring features of, 11–21
- uses folders, 11–18

mhl command in MH, 11–20

mhmail command in MH, 11–20

mkdir command, 3–2, 4–1

mkdir subcommand (ftp), 12–7

monetary formats, C–5

- representation of fractions, C–5

monitoring UUCP, 14–32

more command, 3–6
used by mail to display messages, 11–17,
11–8

more than one word in command argument,
1–11

move command, 4–10

move (m) subcommand (ed editor), B–19

moving
files, 3–22, 3–23
text
ed editor, B–19
vi text editor, A–17

moving directories, 4–10

mput subcommand (ftp), 12–5

msgcheck command in MH, 11–20

multibyte characters
support for in codesets, C–2

multiple names for same file, 3–19

multitasking, 6–6

mv command, 3–22, 4–10, 3–23

N

naming files, 2–6

next command in MH, 11–20

noignore environment variable, A–23

nonumber environment variable, A–23

noshowmatch environment variable, A–23

number environment variable, A–23

numeric formats, C–5

O

octal numbers
in setting permissions, 5–11

online documentation, 1–9

open line (o) command (vi text editor), A–11

open previous line (O) command (vi text editor), A–11

open subcommand (ftp), 12–5

options to commands, 1–5

options, mail
specifying interactively, D–1

owner
changing for files and directories, 5–19

P

packf command in MH, 11–20

page (pg) command, 3–6

parameter substitution, 7–21

parent directory, 2–10

parentheses
using to group commands, 7–11

passwd command, 1–6

password
file, 5–1, 5–2
for logging in, 1–2
forgotten, 1–9
restrictions, 1–8
security restrictions, 1–7
selecting new, 1–7
setting, 1–6
setting with enhanced security system, 1–1

PATH environment variable, 7–15
setting for System V habitat, 9–2

PATH variable, 7–23

path, modifying to use MH program, 11–17

pathname, 2–10, 2–8
absolute, 2–10
dot notation, 2–11
full, 2–10
relative, 2–10

pathname (cont.)

using tilde in, 2–12

pathname conventions (UUCP), 14–1

pattern matching, 2–13

changing file permissions with, 5–10

internationalized, 2–14

list of allowable characters, 2–14

removing multiple files, 3–28

permission combinations for umask, 5–14

permissions

binary numbers, 5–12

changing, 5–3

combinations, 5–12

directory, 5–5

read, 5–9

setting absolute, 5–11

setting file and directory, 5–8

setting with octal numbers, 5–11

specifying with umask, 5–15

pg command, 3–6

pick command in MH, 11–20

PID number, 6–7

pipe character, 7–8

pipes and filters

running multiple commands, 7–7

POSIX locale, C–7

pr command, 3–7

prev command in MH, 11–20

prevent erasure of files, 3–20

print

working directory (pwd), 2–7

print (p) subcommand (ed editor), B–3

printer

specifying in print command, 3–10

printer queues, 3–10, 3–12

printing

files, 3–10

on a specific printer, 3–11

on default printer, 3–11

options, 3–11

reference pages, 1–9

Process Identification Number (PID), 6–7

process identifier, 6–1

processes, 6–1

displaying active, 6–16

displaying status, 6–8, 6–9

displaying who is running them, 6–14

grouping commands, 7–11

killing, 6–12

resuming, 6–13

running through pipes, 7–10

stopping, 6–11

.profile login script

modifying to use MH program, 11–18, 7–17,

8–14, 8–20

modifying for System V Habitat, 9–2

programs

types of, 6–1

prompt

question mark as, in the mail program, 11–7

prompter command in MH, 11–20

prompts for shells, 7–5

protecting files, 5–3

ps command, 6–8, 6–9

public directory (UUCP), 14–1

put subcommand (ftp), 12–5

pwd command, 2–7

Bourne shell, 8–19

Korn shell, 8–34

pwd subcommand (ftp), 12–7

Q

question mark

as mail prompt, 11-7

queues

printer, 3-10

quit (q) command (vi editor), A-4

quit (q) command (vi text editor), A-16

quit (q) subcommand (ed editor), B-6

quit subcommand (ftp), 12-5

quit subcommand (telnet), 13-4

quotes

double, 7-13

single, 7-12

quoting, 7-11

backslash, 7-12

double quotes, 7-13

single quotes, 7-12

to display variable names, 7-12

R

R command in mail, 11-10

r command in mail, 11-10

r (read) permission, 5-9

rcvstore command in MH, 11-20

read permission, 5-5

read (r) subcommand (ed editor), B-6, B-8

reading input, 6-2

reading input from pipes, 7-8

reading mail messages, 11-7

by number, 11-8

reading messages in MH, 11-18

receiving files (UUCP), 14-21

record variable for mail, 11-17

recv subcommand (ftp), 12-5

redirecting

errors, 6-4

input, 6-2

output, 3-27, 6-2, 6-3

output, of background processes, 6-7

redirecting both standard errors and output,
6-5

redirecting errors, 6-4

Bourne shell, 6-4

C shell, 6-5

Korn shell, 6-4

redirecting input/output, 6-1

reexecuting commands, 8-26, 8-9

reference pages, 1-9

referencing variables, 7-20, 7-21

refile command in MH, 11-20

rehash command (C shell), 8-13

relative pathname, 2-10

remote commands, running in UUCP, 14-14

remote file transfers (UUCP), 14-14

remote host

running commands from (UUCP), 14-18

remote login, 13-1, 13-3

remove directory command, 4-12

removing

absolute permissions, 5-11

characters

ed editor, B-16

vi text editor, A-13

current directory, 4-12

directories, 4-10, 4-11, 4-12

file links, 3-17

files, 3-27, 3-28

jobs from print queue, 3-13

links, 3-17

- removing files**
 - with verification, 3–29
- rename subcommand (ftp), 12–7**
- renaming**
 - files and directories, 3–22
- renaming directories, 4–10**
- repeat command (C shell), 8–13**
- replacing character strings**
 - ed editor, B–13
 - vi text editor, A–20
- reply command in MH, 11–20**
- replying to mail messages, 11–10**
 - caution when, 11–10
 - including other recipients in your reply, 11–10
- reset environment variables, 7–21**
- restart a process, 6–13**
- Restricted Bourne shell, 7–1**
- restricted Bourne shell, 7–4**
- restricted characters in file names, 2–6**
- restricting file access, 5–15**
- restricting user environment**
 - restricted Bourne shell, 7–4
- restrictions**
 - password, 1–8
- resuming a process, 6–13**
- returning to local host during remote connection (UUCP), 14–14, 14–8**
- rlogin command, 13–1**
- rm command, 3–17, 3–27, 3–28**
- rmdir command, 4–10, 4–11, 4–12**
- rmdir subcommand (ftp), 12–7**
- rmf command in MH, 11–20**
- rmm command in MH, 11–20**
- root directory, 2–9**

- root user**
 - becoming, 5–19
 - defining shell for, 5–19
 - tasks performed by, 5–18
- run shell, 7–28**
- runique subcommand (ftp), 12–5**
- running**
 - background processes, 6–6, 6–7
 - commands conditionally, 7–8
 - commands in sequence, 7–7
 - foreground processes, 6–6
 - shell procedures, 7–27, 7–28
- running commands on a remote host (UUCP), 14–18**
- runtime command, 10–5**

S

- s (set) permission, 5–9**
- saving**
 - text files in vi, 2–3
- saving mail messages**
 - in files, 11–13
 - in folders, 11–13
- saving part of a file**
 - ed editor, B–5
 - vi text editor, A–21
- saving text**
 - ed editor, B–2, B–4, B–5
 - vi text editor, A–16
- scan command in MH, 11–20**
- security**
 - enhanced system, 1–1
 - group, 5–2
 - restricted Bourne shell, 7–4
 - user, 5–2

security considerations, 5–20

security files, 5–1

selecting a folder in mail, 11–13

semicolon

running commands in sequence, 7–7

send command in MH, 11–20

send subcommand (ftp), 12–5

sending

files to remote hosts (UUCP), 14–14

files via UUCP, 14–21

local messages, 11–21

mail messages, 11–2

sending a long message, 12–9

sending mail messages, 11–3

set command

Bourne shell, 8–19

C shell, 8–13

Korn shell, 8–34

set user/group ID permission, 5–9

setenv command, 7–20

setenv command (C shell), 8–13

setting

absolute permissions, 5–11

aliases in C and Korn shells, 7–3

default file permissions, 5–13

environment variables, 7–14, 7–18

environment variables in all shells, 7–21

environment variables in C shell, 7–20

file and directory permissions, 5–8

file permissions, 5–3

PATH variable, 7–24

permissions using octal numbers, 5–11

user mask, 5–13

setting password

with enhanced security system, 1–1

shell

aliases, 8–10, 8–30

assign default values to environment, 7–13

built-in commands, 8–12, 8–18, 8–33

built-in variables, 8–11, 8–17, 8–32

changing, 7–4

changing permanently, 7–6

changing temporarily, 7–5

command history, 8–25, 8–8

command interpreter, 1–4

comparison of features, 8–2

default prompts, 7–5

default run shell, 7–28

default shell, 7–1

defining custom variables, 7–19

defining for root user, 5–19

editing command lines, 8–27

features, 8–1

filename completion, 8–29, 8–9

login script, 8–14, 8–20, 8–22

mailing files from, 11–6

metacharacters, 8–16, 8–24, 8–6

program, 1–3

prompt, 1–2

restricting users, 7–4

sample C shell login script, 8–3

scripts, 7–26

setting environment variables, 7–21

using the source command, 11–18e

special characters, 2–6, 7–11

subshells, 7–2

SHELL environment variable, 7–15

shell scripts, 7–26

compatibility with System V, 9–4

writing, 7–26

shell variables, 7–16
 defining custom, 7–19
show command in MH, 11–20
showmatch environment variable, A–23
single quotes, 7–12
soft links, 3–14
sort command, 3–26
sort file contents, 3–26
sort rules, C–3
sorting file contents, 3–26
sortm command in MH, 11–20
source command
 in C shell, 8–13
 to invoke shell options for MH, 11–18e
special characters, 7–11
specifying
 default run shell, 7–28
 files with pattern matching, 2–13
 printer in print command, 3–10
standard error, 6–1, 6–4
standard input, 6–1, 6–2
 filtering, 7–9
standard output, 6–1, 6–2
starting the ed editor, B–2
starting the vi text editor, A–4
status information (UUCP), 14–27
status subcommand (ftp), 12–8
status subcommand (telnet), 13–4
stderr, 6–2
stdin, 6–2
stdout, 6–2
stop and restart a process, 6–13
stopping a process with Ctrl/C, 6–11
stopping commands, 1–6
stopping (killing) a job or process, 6–12

structure of directories, 2–8
su command, 5–17
subcommands (ftp), 12–5, 12–7, 12–8
subdirectories, 2–7
subject
 entering for a mail message, 11–16, 11–3
 getting a prompt for in mail, 11–16
subshells, 7–11, 7–2
substituting
 parameters, 7–21
substitute (s) subcommand (ed editor), B–13
substitution
 global, in vi text editor, A–20
sunique subcommand (ftp), 12–5
superuser, 5–17
superuser authority, 5–5
 tasks, 5–18
suspend a process, 6–13
switch to root user, 5–19
symbolic links, 3–14
 and inode numbers, 3–17
system administrator responsibilities, 5–18
system prompts, 7–5
System V habitat, 9–1 to 9–5
 accessing, 9–2
 command summary, 9–4 to 9–5
 location of, 9–3
 modifying .cshrc file for, 9–2
 modifying .login file for, 9–2
 modifying .profile file for, 9–2

T

tabstop environment variable, A–23
talk command
 use of CRT screen by, 11–24

tasks performed by root user, 5–18

telephone number, specifying with cu command (UUCP), 14–5

telnet command, 13–3
how to use, 13–3

TERM environment variable, 7–15

terminating a connection (local communications), 11–22, 11–23

terminating a job or process, 6–12

terminating a UUCP job with the uustat command, 14–27

terminating remote cu connection (UUCP), 14–8

terminating remote tip connection (UUCP), 14–14

territory
part of locale, C–2

text editors
overview of, 2–1

tilde
in text files, 2–2
notation in pathname, 2–12

tilde as escape character in mail, E–1

time command (C shell), 8–13

time format, C–4
punctuation in, C–4

times command
Bourne shell, 8–19
Korn shell, 8–34

tip command (UUCP)
options, connecting to a remote computer, 14–8
options, connecting to a remote host, 14–9
using local commands, 14–12, 14–9

touch command, 3–29

transfer (t) subcommand (ed editor), B–23

transfer-status information (UUCP), 14–27

trap command
Bourne shell, 8–19
Korn shell, 8–34

tree structure, 2–8

tree structure (file system), 2–5

types of programs, 6–1

typing errors, correcting
ed editor, B–3
vi text editor, A–11

TZ environment variable, 7–15

U

umask
permission combinations, 5–14

umask command, 5–13
Bourne shell, 8–19
C shell, 8–5
Korn shell, 8–34

unalias command
C shell, 8–13
Korn shell, 8–34

undo (u) command (vi text editor), A–15

UNIX
case sensitive, 2–6

UNIX-to-UNIX Copy Program (UUCP), 14–1

unset command, 7–23
Bourne shell, 8–19
C shell, 8–13
Korn shell, 8–34

unsetenv command, 7–23

unsetenv command (C shell), 8–13

uppercase characters, 2–6

user commands
summary, 9–5

- user environment**, 7–13
 - assign default values, 7–13
- user identity**
 - confirming, 5–18
- user mask**, 5–13
 - activating in login script, 5–16
 - default, 5–17
- username**, 1–2
- users**
 - displaying inactive, 6–15
 - displaying who is logged in, 6–14
- using**
 - backslash to quote a single character, 7–12
 - braces to group commands, 7–11
 - commands, 1–4
 - delete key to correct mistakes, 1–5
 - filters, 7–8
 - parentheses to group commands, 7–11
 - pipelines, 7–8
 - pipes and filters to run multiple commands, 7–7
 - tilde in pathname, 2–12
 - wildcards in filenames, 2–13
- /usr/spool/uucppublic file (UUCP)**, 14–1
- UUCP**, 14–1
- uulog command (UUCP)**, 14–31
- uumonitor command (UUCP)**, 14–32
- uustat command (UUCP)**, 14–27
- uuto command (UUCP)**
 - copying files, local host control, options, 14–24
- uux command (UUCP)**, 14–18
 - options, used to run remote commands, 14–14

V

- variable name**
 - displaying, 7–12
- variables**
 - clearing values of, 7–22
 - displaying values for, 7–22
 - in mailx, D–1
 - mail, 11–16 to 11–17
 - referencing, 7–20
 - shell built-in, 8–11
- variables, shell built-in**, 8–17, 8–32
- verbose subcommand (ftp)**, 12–8
- vi command**, 2–2
- vi editor**
 - \$ cursor movement command, A–8
 - (cursor movement command, A–8
 -) cursor movement command, A–8
 - { cursor movement command, A–8
 - } cursor movement command, A–8
 - 0 cursor movement command, A–8
 - b cursor movement command, A–7
 - command mode, A–6
 - Ctrl/B cursor movement command, A–8
 - Ctrl/F cursor movement command, A–8
 - h cursor movement command, A–6
 - j cursor movement command, A–6
 - k cursor movement command, A–6
 - l cursor movement command, A–6
 - used to edit mail messages, 11–4
 - moving within a file, A–6
 - quit (q) command, A–4
 - w cursor movement command, A–7
- vi environment variables**
 - errorname, A–23
 - ignorecase, A–23
 - noignorecase, A–23

vi environment variables (cont.)

- nonumber, A-23
- noshowmatch, A-23
- number, A-23
- showmatch, A-23
- tabstop, A-23
- wrapmargin, A-23
- wrapscan, A-23

vi text editor

- / search command, A-16
- add (a) command, A-10
- append text (A) command, A-11
- change (c) command, A-14
- change word (cw) command, A-14
- clearing a line (D) command, A-14
- context searching, A-16
- copying text, A-18
- correcting typing errors, A-11
- customizing your environment, A-22
- delete character (x) command, A-13
- delete line (dd) command, A-13
- delete word (dw) command, A-13
- deleting a block of text, A-22
- environment variables, A-23
- ex line editor commands, A-18
- getting started, A-2
- insert (i) command, A-10
- insert text (I) command, A-11
- locating text, A-16
- map command, A-23
- moving text, A-17
- next (n) search command, A-17
- open line (o) command, A-11
- open previous line (O) command, A-11
- opening text files, A-4
- paste (p) command, A-17

vi text editor (cont.)

- quit (q) command, A-16
- saving part of a file, A-21
- saving text files, A-16
- saving your customizations, A-24
- searching for text, A-16
- starting the editor, A-4
- substituting text, A-20
- undo (u) command, A-15
- using advanced techniques, A-16
- write (w) command, A-16

virus, 5-21

W

w command, 6-15

w (write) permission, 5-9

whatnow command in MH, 11-20

who command, 6-14

who is logged on, 1-10

whoami command, 5-17

- confirming identity, 5-18

whom command in MH, 11-20

wildards

- removing multiple directories, 4-12

wildcards, 2-13

- changing file permissions with, 5-10

- use in removing files, 3-29

working directory, 2-7

wrapmargin environment variable, A-23

wrapscan environment variable, A-23

write command (local communications),

- 11-21

write permission, 5-5, 5-9

write (w) command (vi text editor), A-16

write (w) subcommand (ed editor), B-4, B-6

writing

logout script, 7–24

mail messages, E–1

shell scripts, 7–26

shell scripts (example), 7–27

X

x (execute) permission, 5–9

Z

z subcommand (telnet), 13–4

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825) before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-bps modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECDirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	—————	Local Digital subsidiary or approved distributor
Internal ^a	—————	SSB Order Processing – NQO/V19 <i>or</i> U. S. Software Supply Business Digital Equipment Corporation 10 Cotton Road Nashua, NH 03063-1260

^a For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

Digital UNIX
Command and Shell User's Guide
AA-PS2HD-TE

.....

Digital welcomes your comments and suggestions on this manual. Your input will help us to write documentation that meets your needs. Please send your suggestions using one of the following methods:

- This postage-paid form
- Internet electronic mail: `readers_comment@zk3.dec.com`
- Fax: (603) 881-0120, Attn: UEG Publications, ZKO3-3/Y32

If you are not using this form, please be sure you include the name of the document, the page number, and the product name and version.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)
Completeness (enough information)
Clarity (easy to understand)
Organization (structure of subject matter)
Figures (useful)
Examples (useful)
Index (ability to find topic)
Usability (ability to access information quickly)

Please list errors you have found in this manual:

Page	Description
------	-------------

.....
.....
.....
.....
.....

Additional comments or suggestions to improve this manual:

.....
.....
.....
.....
.....

What version of the software described by this manual are you using?

Name/Title Dept.

Company Date

Mailing Address

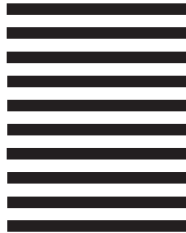
..... Email Phone

----- Do Not Cut or Tear – Fold Here and Tape -----

digitalTM



NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
UEG PUBLICATIONS MANAGER
ZK03-3/Y32
110 SPIT BROOK RD
NASHUA NH 03062-9987



----- Do Not Cut or Tear – Fold Here -----

Cut on
Dotted
Line